VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rozšíření nástroje pro řízení projektů o fulltextové vyhledávání

Project Management Tool Extension with an Fulltext Searching Capability

2010

Martin Šustek

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 .....

Abstrakt

Cílem této práce je doplnit nástroj Project.net, který slouží pro řízení projektů, o fulltextové vyhledávání v dokumentech. Tento nástroj sice již umožňuje uživatelům nahrávat do systému dokumenty, vyhledávání v nich se však omezuje pouze na název a popis.

Výsledkem práce je propojení aplikace Project.net s databázovým serverem MSSQL 2008, který poskytuje funkce fulltextového vyhledávání v obsahu dokumentů v různých textových i binárních formátech. Zároveň je cílem co nejjednodušší nasazení výsledného řešení.

Klíčová slova: Project.net, fulltextové vyhledávání, binární dokumenty, Oracle Database, MSSQL 2008, Java, .NET

Abstract

The goal of this thesis is to improve project management tool called Project.net by full-text search in documents. This tool already have ability to store user documents, but search is limited only to document title and description.

The result of entire work is connection between Project.net application and MSSQL 2008 database server, which serves full-text search in various text and binary file formats of documents. Also the goal is to make installation of whole solution as easy as possible.

Keywords: Project.net, full-text search, binary documents, Oracle Database, MSSQL 2008, Java, .NET

Seznam použitých symbolů a zkratek

CSS	–	**Cascading Style Sheets** – jazyk definující vzhled (styl) HTML dokumentu, více v kapitole 2.1.2.3.
DBMS	–	**DataBase Management System** – množina programů a služeb pro správu databáze, nebo také databázový server, například MSSQL 2008 nebo Oracle Database apod.
GUI	–	**Graphical User Interface** – uživatelské rozhraní využívající grafických prvků pro usnadnění obsluhy počítačových programů.
HTML	–	**HyperText Markup Language** – značkovací jazyk, používaný pro popis strukturovaných dokumentů, převážně v internetovém prostředí, tedy na webu. Vizte také kapitolu 2.1.2.2.
HTTP	–	**HyperText Transfer Protocol** – protokol umožňující přenos hypertextu, a obecně jakýchkoli souborů přes internet. Více v kapitole 2.1.2.1.
JDBC	–	**Java DataBase Connectivity** – jednotné rozhraní pro práci s relačními databázemi v jazyce Java
JDK	–	**Java Development Toolkit** – sada nástrojů pro vývoj programů v jazyce Java
JSP	–	**JavaServer Pages** – technologie pro tvorbu obsahu webových stránek na serveru. Podrobnosti v kapitole 2.1.2.6.
MSSQL 2008	–	**Microsoft SQL Server 2008**
SVN	–	**Subversion** – nástroj pro správu zdrojových kódů, mj. také zdrojových kódů aplikace Project.net. Podrobnosti naleznete v kapitole 2.1.2.9
URL	–	**Uniform Resource Locator** – adresa identifikující umístění dokumentu v rámci internetu. URL je ta adresa, která se zadává do webového prohlížeče.
W3C	–	Konsorcium, které se zabývá webem a vydává standardy popisující webové technologie.

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

Seznam tabulek

1 Úvod

Project.net je open source nástroj pro management projektů. Oficiální webová stránka programu je http://www.project.net/ [1].

1.1 Výchozí stav aplikace a cíl úprav

Aplikace Project.net umožňuje uživatelům ukládat a organizovat dokumentaci. Dokumenty jsou ukládány v centrálním úložišti zvaném *Document Vault* ve formě souborů, každý dokument vždy ke konkrétnímu objektu – uživateli, projektu apod.

Aplikace dále obsahuje jednoduché rozhraní pro vyhledávání dokumentů. Toto rozhraní ale umí prohledávat pouze názvy a popisky dokumentů, a nemá žádnou možnost prohledávat obsah dokumentů. Dalším problémem je, že vyhledávat je možné pouze v aktuálně otevřené agendě, tedy například pouze v jednom projektu.

Cílem této diplomové práce je doplnit správu dokumentů o fulltextové vyhledávání v obsahu dokumentů.

Řešení představené v této práci umožňuje prohledávat text uložený v textových souborech i binárních dokumentech, tedy souborech typu TXT, HTML, PDF, DOC, ODT, XLSX a dalších více jak 150 formátech dokumentů.

Při formulaci vyhledávacích dotazů je možné použít základní logické spojky. Dále je přidána možnost prohledávat úplně všechny dokumenty, tedy nejen aktuálně otevřenou agendu.

Přitom je ovládání upravené aplikace stejně jednoduché jako původní vyhledávání. Důraz je kladen také na jednoduchou instalaci celého řešení.

1.2 Obsah tohoto textu

V první části této práce představím samotný systém Project.net, a způsob, jak se v něm ukládají dokumenty uživatelů. Dále věnuji část textu technologiím, na kterých je aplikace vystavěna, jež je třeba pro provádění úprav znát.

Druhá kapitola ukáže Microsoft SQL Server (MSSQL), a pojedná o tom, jak jej lze využít pro ukládání a fulltextovou indexaci dokumentů, a také popíše vyhledávací dotazy, které lze nad těmito strukturami formulovat. Technika představená v tomto textu je obecně použitelná i pro jiné aplikace, a proto se snažím vyhledávání popsat nejprve obecně.

Ve třetí části jsou už konkrétní praktické ukázky a poznatky z návrhu a implementace fulltextu do Project.net. Je zde popsán i program pro rychlou konfiguraci celého řešení. Jednou z příloh této práce je i ukázkové video, demonstrující vyhledávání v akci.

V závěru zhodnotím, jak se podařilo splnit požadavky kladené na vyhledávání a shrnu možnosti dalšího rozšíření této práce a využití jejich výsledků.

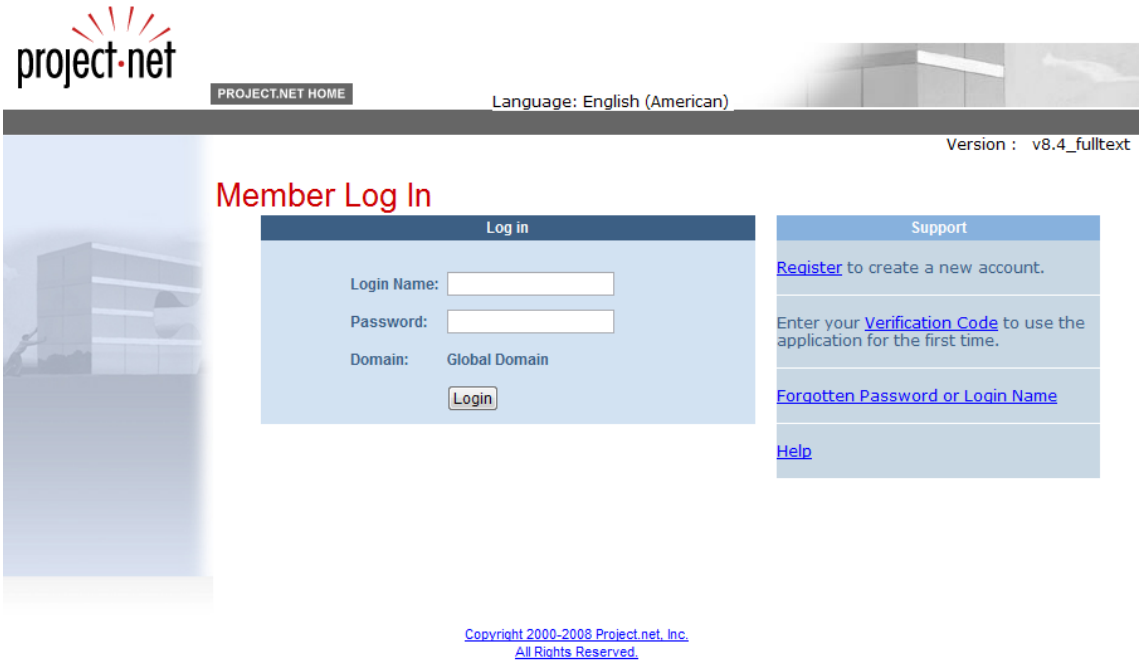
2 Seznámení s nástrojem Project.net

V této kapitole popisuji aplikaci Project.net, její aplikační architekturu a klíčové technologie, na nichž je postavena. Pro provádění úprav aplikace je potřeba ovládat minimálně základy těchto technologií.

Dále v této kapitole vysvětluji, jakým způsobem lze získat zdrojové kódy aplikace a ukazuji postup, jímž si lze připravit vývojové prostředí pro sestavení aplikace.

Na zkompilované a spuštěné aplikaci předvedu základní nastavení, a poté už bude možné vyzkoušet si práci s dokumenty, ještě ovšem bez fulltextového hledání.

Poslední část kapitoly popisuje, jak správa dokumentů v aplikaci Project.net funguje, a jaké akce s dokumenty lze provádět.



Obrázek 1: Přihlašovací stránka aplikace Project.net

2.1 Popis aplikace Project.net

Project.net je webová aplikace, z převážné části napsaná v jazyce Java. Provozovat ji lze na webovém serveru Apache Tomcat[1] nebo Oracle WebLogic[2].

Pro tuto práci jsem si zvolil server Apache Tomcat 5.5, který je uváděn jako doporučený na webovém portálu pro vývojáře Project.net [2].

Databázovým serverem pro Project.net je Oracle, já jsem pracoval s Oracle Database 10g Express Edition. Tato verze je k dispozici ke stažení zdarma[3].

2.1.1 Verze aplikace 8.4

Po dohodě s vedoucím práce jsem se rozhodl vyjít ze starší verze aplikace Project.net, tedy verze 8.4. V současné době je sice k dispozici verze 9.1, s touto verzí však na začátku neměl ani jeden z nás žádné zkušenosti.

Navíc instalace nové verze byla velmi problémová. Aplikace totiž prošla mezi těmito verzemi poněkud bouřlivějším vývojem, a dostupná dokumentace byla v mnoha ohledech neaktuální, resp. se jednalo o dokumentaci pro starší verzi.

Starší verze 8.4 už byla ověřená, běžela na několika instalacích bez problémů a nebyl žádný další důvod z ní přecházet.

Kdybych připravil fulltext pro novou verzi, byli bychom omezeni na ni, a stávající běžící systémy bychom museli přeinstalovat. Instalace nové verze zatím nebyla nutná, a mohla by být zdrojem potenciálních problémů. Takže jsme se rozhodli pro konzervativnější variantu – zachování starší verze.

2.1.2 Použité technologie a architektura aplikace

Z technologického hlediska je aplikace poněkud zastaralá. Sami autoři to přiznávají [2], a zdůvodňují faktem, že v době, kdy se psaly základy aplikace, ještě nebyly k dispozici moderní aplikační frameworky typu Spring.

Na několika následujících stránkách se pokusím vyjmenovat a blíže popsat technologie a principy, které jsou v aplikaci Project.net využívány.

[1] http://tomcat.apache.org/
[2] http://www.oracle.com/technology/products/weblogic/
[3] http://www.oracle.com/global/cz/database/express_edition.html

2.1.2.1 HyperText Transfer Protocol (HTTP) Pokud chápeme internet, jako množinu stránek, rozprostřenou po celém světě, potřebujeme mechanismus, jak přenést konkrétní stránku z konkrétního serveru ke konkrétnímu uživateli. Tímto mechanismem je protokol HTTP.

Řečeno terminologií referenčního modelu ISO/OSI: HTTP je internetový protokol na aplikační vrstvě. Jedná se o protokol, popisující komunikaci mezi webovým klientem a serverem. Tato komunikace probíhá v textové formě.

Komunikace vždy začíná na straně klienta, který odešle požadavek, zvaný HTTP request, na server. Požadavek může vypadat například takto:

```
1 GET /index.html HTTP/1.1
2 Host: www.project.net
3 User−Agent: Mozilla/5.0 Gecko/20040803 Firefox/0.9.3
4 Accept−Charset: UTF−8,∗
```

Výpis 1: Ukázka HTTP požadavku

Požadavek obsahuje hlavičky definující jaký dokument chceme stáhnout, obsahuje i informace o použitém webovém prohlížeči a podporované typy obsahu, aby server věděl, v jakém formátu má data vrátit. V případě potřeby mohou být zahrnuty i další volitelné informace.

Požadavek přijatý od klienta webový server zpracuje a vrátí zpět odpověď, tzv. HTTP response, která se skládá z informačních hlaviček a těla požadovaného dokumentu:

```
1 HTTP/1.0 200 OK
2 Date: Fri , 15 Oct 2004 08:20:25 GMT
3 Server: Apache/1.3.29 (Unix) PHP/4.3.8
4 X−Powered−By: PHP/4.3.8
5 Vary: Accept−Encoding,Cookie
6 Cache−Control: private, s−maxage=0, max−age=0, must−revalidate
7 Content−Language: cs
8 Content−Type: text/html; charset=utf−8
9
10 <obsah dokumentu>
```

Výpis 2: Ukázka HTTP odpovědi

První hlavička popisuje použitou verzi protokolu, a stavový kód. Stavový kód se skládá z třímístné číslice a textové informace. Seznam vybraných kódů je v tabulce 1.

Ostatní HTTP hlavičky popisují typ vráceného dokumentu, datum poslední aktualizace a další informace pro prohlížeč, například o tom, jakým způsobem s dokumentem nakládat, jestli je možné dokument uchovávat v lokální cache apod.

Vzhledem k tomu, že HTTP je nešifrovaný textový protokol, není složité komunikaci odposlouchávat. Odposlouchávání síťové komunikace může být problém například pro přenos přihlašovacích údajů, nebo jiných citlivých informací. Proto byl vytvořen protokol HTTPS, který je nadstavbou na HTTP a řeší šifrování komunikace.

Aplikace Project.net používá ve standardním nastavení protokol HTTP, ale lze ji nakonfigurovat i pro HTTPS.

HTTP kód	význam
200	OK
301	Moved Permanently
302	Found
304	Not Modified
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
500	Internal Server Error
501	Not Implemented
505	HTTP Version Not Supported

Tabulka 1: Vybrané stavové kódy HTTP protokolu (podle [5])

2.1.2.2 HyperText Markup Language (HTML, XHTML) HTML je jazyk pro popis obsahu webových stránek. Jedná se o textový soubor, jenž je doplněn o speciální značky, tzv. tagy, které dávají jednotlivým částem textu určitý význam, a popisují tak strukturu dokumentu. Proto se o jazyku HTML mluví jako o značkovacím (markup) jazyku.

```
1  <HTML>
2    <HEAD>
3      <TITLE>HTML sample − title</TITLE>
4    </HEAD>
5    <BODY>
6      <H1>Heading</H1>
7      <P CLASS="normal">Paragraph text.</P>
8      <!−− sample comment −−>
9    </BODY>
10 </HTML>
```

Výpis 3: Ukázka HTML kódu

Jednou z hlavních výhod HTML, a důvodem, proč se mu říká hypertext, je možnost vkládat do dokumentu odkazy na jiné části textu. Uživatel pak jednoduše kliknutím na odkaz přejde na jinou stránku, která ho zajímá. Dokumenty je možné rozšiřovat také o obrázky, videa a další objekty.

Vzájemným propojením mezi dokumenty vzniká síť, které se říká World Wide Web, nebo taky zkráceně www nebo web.

Důležitou vlastností HTML je možnost tvořit formuláře. Pokud stránka obsahuje formulář, a uživatel jej vyplní, webový klient ho může odeslat na server prostřednictvím dalšího HTTP požadavku. Díky vzájemné komunikaci klient-server lze vytvářet interaktivní webové aplikace.

Původně jazyk HTML vznikl odvozením z jazyka SGML (Standard Generalized Markup Language), který představuje jakýsi obecný značkovací jazyk. Z jazyka SGML vznikl později jazyk XML (Extensible Markup Language), který je jednodušší a lépe se tedy strojově zpracovává. Z jazyka XML vznikl také jazyk XHTML, který je modernější alternativou za HTML a v současné době se používá na většině webových stránek.

Aplikace Project.net je psána ještě „postaru“ v HTML, nikoli XHTML.

2.1.2.3 Cascading Style Sheets (CSS)

Specifikace HTML obsahuje značky sloužící pro formátování textu a definici vzhledu stránek – volbu velikosti a tvaru písma, barev, pozice objektů na stránce a další. Dnešním trendem je ale oddělení vzhledu stránek od obsahu. Podle tohoto dogmatu by tedy HTML dokumenty měly popisovat pouze sturkturu a obsah textu, zatímco vzhled se má řešit připojením kaskádových stylů (CSS).

Díky tomu, že je možné ukládat obsah odděleně od vzhledu, tedy uložit styly mimo hlavní dokument, je dosaženo znovupoužitelnosti souborů se styly. Můžeme tak mít pro více HTML dokumentů společný CSS soubor, ten se načte pouze jednou, uloží se do cache prohlížeče, a při dalším prohlížení webu se již nestahuje. Tím se urychlí načítání stránek.

Funguje to i naopak – k jednomu HTML dokumentu můžeme přiřadit několik stylů, a tím zajistit jiné zobrazení na monitoru počítače než na tiskárně nebo dataprojektoru.

Pokud chceme změnit vzhled celého webu, stačí nám jen upravit soubor se styly. Změna vzhledu se projeví u všech stránek. Další výhodou CSS je, že poskytuje mnohem větší rozsah možností formátování stránek, než klasické HTML.

Princip kaskádových stylů spočívá v tom, že je vytvořen předpis, podle kterého se (X)HTML kód formátuje. Jednotlivé elementy se vybírají pomocí tzv. selektorů, a vzhled se jim nastavuje pomocí vlastností.

Selektor je textový řetězec, který pomocí typu objektu (body, h1, div, p, span), jeho identifikátoru (id), třídy (class) a dalších vlastností (hodnoty atributů, pořadí elementu ve svém rodiči apod.) definuje množinu HTML elementů, na kterou se styly aplikují. Selektory lze řetězit a vybírat tak elementy hierarchicky (kaskádově).

```
1 h1 {
2     font-size: xx-large;
3     color: red;
4 }
5 div#main p.important {
6     text-decoration: underline;
7     color: blue;
8     border: 2px solid red;
9 }
```

Výpis 4: Ukázka CSS

Aplikace Project.net sice využívá CSS stylů, ale ne všude je vzhled od obsahu plně oddělen.

Vývojem jazyků HTML i CSS se zabývá konsorcium W3C, další informace jsou dostupné na jejich webu [6].

2.1.2.4 Java Java[4] je programovací jazyk, který vyvinula v roce 1995 firma Sun Microsystems (dnes ji vlastní Oracle). V roce 2007 byly zdrojové kódy uvolněny, a jazyk Java je od té doby vyvíjen jako Open Source [7].

Java způsobila jistý převrat na poli programování, a to hlavně tím, že odstranila problematické konstrukce, známé například z jazyků C/C++, jakými byly pointery, a práce s pamětí obecně. Nesprávná práce s pamětí byla hlavní příčinou chyb, bezpečnostních zranitelností a pádů aplikací vytvořených v jazycích, které Javě předcházely.

V jazyce Java se o práci s pamětí stará běhové prostředí, které hlídá, kdy je nutno vyhradit pro program paměť, a kdy je možno ji opět uvolnit. O uvolňování paměti se stará tzv. garbage collector, který monitoruje využití paměti a pomocí sofistikovaných metod rozhoduje, kdy je který úsek paměti nevyužívaný, a je tedy vhodné jej uvolnit. Přitom kalkuluje jak s ohledem na potřebné množství volné paměti, tak i s ohledem na výpočetní čas potřebný k jejímu uvolnění.

Programy v jazyce Java nejsou kompilovány do strojového kódu konkrétního typu počítače, ale do tzv. mezikódu (bajtkódu), tedy kódu jakéhosi obecného počítače. Bajtkód je nezávislý na konkrétní architektuře, a je tedy přenositelný mezi platformami. Na cílové platformě pak kód běží uvnitř tzv. virtuálního stroje Javy, což je vlastně program běžící na hostitelském počítači a umožňující zpracování instrukcí bajtkódu. Zároveň také realizuje ochranu cílového počítače před možnými chybami programu nebo i záměrným poškozením souborů na hostitelském počítači.

Oproti interpretovaným jazykům má tento přístup výhodu v rychlosti, neboť bajtkód je strojovému kódu dnešních počítačů bližší než zdrojový kód, a jeho interpretace za běhu programu je jednodušší. Dalšího zrychlení programů psaných v Javě bylo dosaženo nahrazením interpretace bajtkódu překladem do kódu spustitelného na daném stroji před spuštěním, tzv. Just in Time překlad – JIT. To sice prodlužuje čas potřebný pro spuštění programu, už přeložený program pak ale běží rychleji.

Java je jazyk objektově orientovaný, využívá mechanismu výjimek a dalších moderních prvků programování, díky čemuž jsou programy přehledné a pro programátora lépe srozumitelné. V konečném důsledku je vývoj v Javě rychlejší a vytvořené aplikace kvalitnější. To proto, že pravděpodobnost, že programátor udělá chybu v kódu, který je psán ve srozumitelnějším jazyce Java je nižší, než je tomu například u jazyka C.

Principy využívané v jazyce Java se osvědčily, a využívají je další programovací jazyky, jako například C# nebo J# na platformě .NET.

Většina aplikace Project.net je psána právě v Javě.

[4] http://java.sun.com/

2.1.2.5 Java Servlet Java Servlet[5] je třída napsaná v jazyce Java podle standardů Java Servlet API. Toto API definuje třídy pro popis HTTP požadavku a odpovědi a další podpůrné konstrukce.

Servlet je tedy objekt, jehož úkolem je zpracovat předaný HTTP požadavek, a na jeho základě vytvořit HTTP odpověď, která je vrácena klientovi zpět. Slouží tedy pro programování dynamických webových aplikací.

Na začátku načte webový kontejner třídu servletu, vytvoří její instanci a zavolá na ni metodu *init()*. Pak pro každý HTTP požadavek volá metodu *service()*, která zavolá metodu pro patřičný HTTP požadavek (*doGet()*, *doPost()* apod.). Na konci životního cyklu zavolá webový kontejner na servlet metodu *destroy()*, která se stará o ukončení práce servletu.

Servlety jsou tedy plnohodnotné Java programy, běžící ve virtuálním stroji Javy. Metody jsou volané na základě HTTP požadavků webovým kontejnerem – aplikačním serverem typu Tomcat.

```
1  public class HelloWorld extends HttpServlet {
2      public void doGet(HttpServletRequest req, HttpServletResponse res)
3          throws ServletException, IOException {
4
5          res.setContentType("text/html");
6          PrintWriter out = res.getWriter();
7
8
9          out.println("<HTML>");
10         out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
11         out.println("<BODY>");
12         out.println("<H1>Hello World</H1>");
13         out.println("Today is: " + (new java.util.Date().toString()) );
14         out.println("</BODY></HTML>");
15     }
16 }
```

Výpis 5: Ukázka Servletu [9]

2.1.2.6 JavaServer Pages (JSP) JavaServer Pages[6] je další vývojový stupeň servletů, umožňující snadnější tvorbu HTML stránek. Je to odpověď společnosti Sun na technologie ASP a PHP [8].

Po stránce syntaxe se u JSP jedná, podobně jako u ASP a PHP, o mix HTML, případně XML nebo jiného formátu, se speciálními značkami uvozujícími kód v jazyce Java. Jednotlivé části kódu mohou být „roztrhány“ dalším HTML kódem.

Výsledný kód stránky vznikne kombinací původního HTML kódu a výstupu programu zapsaného mezi ním. Navíc pokud je HTML kód například uvnitř podmínky, vypíše se jen při jejím splnění, obdobně pokud je kód uvnitř cyklu, může se vypsat vícekrát.

[5] http://java.sun.com/products/servlet/
[6] http://java.sun.com/products/jsp/

JSP kód není zpracováván přímo, ale napřed je přeložen do podoby Servletu, který je posléze spuštěn na webovém serveru. Tento překlad se provádí jen jednou, nebo při změně JSP kódu. Překladače JSP jsou typicky součástí aplikačních serverů, například Tomcatu.

Zdrojová JSP stránka, která vygenerovala servlet na předchozím výpisu mohla vypadat například takto:

```
1 <HTML>
2   <HEAD>
3     <TITLE>Hello World</TITLE>
4   </HEAD>
5   <BODY>
6     <H1>Hello World</H1>
7       Today is: <%= new java.util.Date().toString() %>
8   </BODY>
9 </HTML>
```

Výpis 6: Ukázka JSP stránky [9]

V JSP je psána prezentační vrstva aplikace Project.net.

2.1.2.7 Apache Ant Správně sestavit celou aplikaci je relativně složitý úkol, proto tvůrci vytvořili skripty pro program ant[7], který značně usnadňuje sestavení aplikace, což se hodí hlavně při ladění, kdy je třeba kompilovat jednotlivé části kódu velmi často.

Program ant na základě konfiguračního souboru provede všechny úkoly potřebné pro kompilaci aplikace a její instalaci na server. Zároveň kontroluje splnění počátečních podmínek a závislostí mezi úkoly.

I přes použití programu ant je sestavení časově náročné, na mém počítači trvalo zkompilování celého programu, příprava ostatních souborů a nahrání na běžící Tomcat server okolo 2 minut, takže se vyplatilo počínat si během psaní kódu obezřetně, aby nebylo nutné kompilovat aplikaci zbytečně.

2.1.2.8 Eclipse Hlavním nástrojem programátora je editor zdrojového kódu. První programy byly psány v obyčejných textových editorech. Specializované editory postupně začaly umět zvýrazňovat syntaxi, případně automaticky odsazovat a formátovat vnořené bloky kódu. Novější generace editorů umí analyzovat zdrojový kód, nabízet a doplňovat názvy objektů, vlastností, funkcí apod.

Dobré editory, k jakým Eclipse[8] podle mého názoru patří, umí zpracovávat závislosti a vztahy mezi jednotlivými soubory a kontrolovat syntaktickou a sémantickou správnost kódu. Tím značně urychlí práci programátora, neboť ten chyby odhalí už během psaní kódu, a nemusí aplikaci tak často překládat, spouštět a testovat.

Autoři Project.net prostředí Eclipse doporučují, a také pro něj dodávají konfigurační skripty, jak bude uvedeno dále v odstavci 2.2.2.2.

[7] http://ant.apache.org/
[8] http://www.eclipse.org/

2.1.2.9 Subversion Zdrojové soubory aplikace je možné získat pomocí nástroje Subversion[9]. Popisovaná verze aplikace 8.4 se nachází na URL adrese
`http://community.project.net/svn/pnet-root/tags/v8.4.0_release`.

Program Subversion (SVN) slouží ke správě zdrojových kódů aplikace. Kromě poslední verze aplikace uchovává i starší verze, takže je možné kdykoli si prohlédnout starší revizi souboru nebo celé aplikace.

Kromě toho SVN usnadňuje distribuci kódu mezi více lidí, a společnou práci několika lidí na jednom kódu. Přestože každý z uživatelů pracuje s lokální kopií souborů, centrální server kontroluje změny, umožňuje je automaticky nebo manuálně slučovat, a kontrolovat zásahy jednotlivých uživatelů do kódu. Centrální správa také usnadňuje zálohování.

2.1.3 Zdrojový kód aplikace

Zdrojový kód aplikace je rozdělen do několika adresářů, jak je patrno z tabulky 2.

2.2 Instalace aplikace Project.net

2.2.1 Příprava vývojového prostředí

Na úvod je třeba nainstalovat si všechny aplikace potřebné pro stažení zdrojových kódů, sestavení a spuštění aplikace.

Jak jsem se zmínil už dříve, rozhodl jsem se nainstalovat aplikaci Project.net na server Tomcat, takže jsem postupoval podle návodu uvedeného na webu pro vývojáře Project.net [2].

2.2.1.1 Klient pro SVN Pro získání zdrojových kódu je potřeba nainstalovat klienta služby SVN. Program je dostupný ve verzi pro příkazový řádek. Toto rozhraní je sice všemocné, ale pohodlněji se pracuje s některou jeho grafickou nástavbou.

Pro Windows je k dispozici grafické rozšíření TortoiseSVN[10], které navíc integruje SVN do průzkumníka Windows. Kromě toho většina programátorských editorů disponuje vlastním rozšířením pro SVN, například pro Eclipse se mi osvědčil plugin Subclipse[11].

2.2.1.2 Java Development Toolkit Pro programování aplikací v jazyce Java je třeba nainstalovat JDK – Java Development Toolkit. Jedná se o sadu programů potřebných pro vývoj Java aplikací, tedy běhové prostředí, kompilátor, debugger, nástroje pro dokumentaci, a další aplikace.

Autoři doporučují JDK verze 1.5, já jsem si nainstaloval poslední verzi – 1.6.0_02, se kterou mi aplikace taky fungovala.

[9] http://subversion.apache.org/
[10] http://tortoisesvn.tigris.org/
[11] http://subclipse.tigris.org/

složka	obsah
core/	hlavní zdrojové kódy programu
bin/	zkompilované soubory pro nasazení na serveru
config/	konfigurační soubory
crypto/	šifrovací klíče a certifikáty
eclipse/	konfigurační soubory pro Eclipse
hibernate/	konfigurace nástroje Hibernate
licence-files/	open-source licenční soubory
log4j/	vzorové konfigurační soubory pro log4j
mvc/	konfigurace MVC – mapování URL adres na handlery
scheduler/	nastavení periodických služeb - cron
spring/	konfigurace Spring
db/	skripty pro instalaci databáze
edb/	EnterpriseDB (v přípravě)
oracle/	instalační skripty pro Oracle
lib/	systémem vyžadované knihovny – soubory `.jar`
src/	zdrojové soubory `.java`
net/project/	zdrojové soubory Project.net
document/	management dokumentů
search/	vyhledávání
org/json/	zdrojové soubory JSON
web/	soubory pro webové rozhraní
css/	kaskádové styly (definice vzhled HTML kódu) – `.css`
html/	HTML šablony – `.html`
images/	obrázky – `.gif`, `.jpg`, `.png`
jsp/	JSP soubory – `.jsp`
META-INF/	konfigurační soubory pro Tomcat/WebLogic
src/	zdrojové soubory java-scriptu – `.js`
WEB-INF/	konfigurační soubory pro Tomcat/WebLogic
xml/	pomůcky pro XML soubory – `.dtd` a `.xsd` schémata
test/	testovací rutiny
acceptance/	akceptační testy – black-box funkční testy
load-test/	zátěžové testy
unit-test/	jednotkové testy
tools/	pomůcky pro sestavení, instalaci a správu
ant/	nástroj pro sestavení a instalaci na webový server

Tabulka 2: Základní struktura zdrojových kódů aplikace

2.2.1.3 Tomcat Autoři v návodu doporučují Tomcat verze 5.5, použil jsem tedy tuto verzi. Během instalace se program dotáže, na kterém portu má server běžet a naslouchat příchozím HTTP spojením. Protože na svém počítači už používám na standardním portu 80 jiný webový server, nastavil jsem Tomcat, aby využíval port 8090.

Dále je třeba v instalátoru zvolit přihlašovací jméno a heslo pro správce serveru a nastavit cestu k nainstalovanému JDK.

2.2.1.4 Ant Nainstalovat nástroj Ant je jednoduché, nevyžaduje žádné pokročilé nastavení. Autoři doporučují verzi 1.6.5.

2.2.1.5 Oracle Database 10g Databázový server Oracle 10g je k dispozici ve verzích Express, Standard a Enterprise, přičemž verze Express je zdarma, a proto jsem použil ji.

Během instalace databáze je třeba pojmenovat její instanci a nastavit správcovské heslo, které bude ještě potřeba pro vytvoření databázových tabulek aplikace Project.net.

2.2.2 Instalace a konfigurace aplikace

Když už máme splněny všechny požadavky, můžeme se pustit do instalace aplikace. Tento postup vychází z návodu na stránkách pro vývojáře Project.net [2].

2.2.2.1 Checkout zdrojových kódů Zdrojové kódy lze stáhnout z SVN repozitáře příkazem checkout – z příkazové řádky:

```
1 svn checkout http :// community.project.net/svn/pnet-root/tags/v8.4.0_release
```

Výpis 7: Příkaz pro checkout souborů z SVN

Já jsem ale použil Eclipse, který ze stažených souborů rovnou vytvořil nový projekt.

2.2.2.2 Konfigurace Eclipse Aby v Eclipse vše správně fungovalo, je podle autorů třeba zkopírovat ze složky *core/config/eclipse* konfigurační soubory *.classpath*, *.project* a složky *.externalToolBuilders* a *settings* do kořenové složky projektu.

Tím by se měl podle návodu Eclipse nakonfigurovat pro správné sestavení aplikace. Nicméně v mém případě to nefungovalo, takže jsem nakonec spouštěl sestavovací skript *core/build-tomcat.xml*, byť z prostředí Eclipse, ručně.

2.2.2.3 Vytvoření databáze Pro správnou funkci aplikace Project.net je třeba ještě vytvořit databázi. K tomu slouží skript *core/db/oracle/createscripts/admin/pnetMasterDBBuild.bat*.

Ke spuštění skriptu je třeba nastavit hodnoty některých proměnných, podrobnosti jsou popsány uvnitř souboru. Po spuštění skriptu se v databázi vytvoří potřebné tabulky a následně se naplní inicializačními daty.

2.2.2.4 Nastavení aplikačního serveru Tomcat

V Tomcatu je třeba založit uživatele a přiřadit jim správná oprávnění.

To se dá udělat editací souboru *conf/tomcat-users.xml* v adresáři, kde je nainstalován Tomcat. Podle návodu by měl soubor vypadat asi takto:

```
1  <?xml version='1.0' encoding='utf-8'?>
2  <tomcat-users>
3      <role rolename="tomcat" />
4      <role rolename="role1" />
5      <role rolename="manager" />
6      <user username="tomcat" password="tomcat" roles="tomcat" />
7      <user username="both" password="tomcat" roles="tomcat,role1" />
8      <user username="role1" password="tomcat" roles="role1" />
9      <user username="manager" password="manager" roles="manager" />
10 </tomcat-users>
```

Výpis 8: Konfigurace serveru Tomcat – *conf/tomcat-users.xml*

Pak je ještě potřeba knihovny *jdbc/ojdbc14.jar*, *mail.jar* a *activation.jar* zkopírovat ze složky *core/lib* Project.net do složky *common/lib* v adresáři s Tomcatem, a soubory *serializer.jar*, *xalan.jar*, *xercesImpl.jar* a *xml-apis.jar* ze složky *core/lib/endorsed/* v Project.net do složky *common/endorsed* v Tomcatu.

Je docela možné, že to není úplně nutné, třeba by stačilo nastavit cestu k těmto souborům v konfiguraci Tomcatu, ale raději jsem se řídil oficiálním v návodem.

2.2.2.5 Konfigurace Project.net

Pro sestavení Project.net je nutná knihovna *catalina-ant.jar*. Stačí ji zkopírovat ze složky *server/lib* v Tomcatu do složky *lib* v adresáři s programem Ant.

Dále je třeba nakonfigurovat soubor *core/build-tomcat.properties* ve složce s Project.net. Je tam i ukázkový soubor *build-tomcat.properties.example*, takže jej stačí vzít za vzor a jen v něm upravit příslušné hodnoty, například takto:

```
17 catalina.home=D:\\Apache Software Foundation\\Tomcat 5.5
18
19 tomcat.url=http://localhost:8090//manager
20 tomcat.username=manager
21 tomcat.password=manager
22
23 tomcat.path=/
```

```
35 tomcat.url=http://localhost:8090//manager
36 tomcat.username=manager
37 tomcat.password=manager
```

```
44 tomcat.path=/
```

Výpis 9: Konfigurace serveru Tomcat – *core/build-tomcat.properties*

Další krok je nastavení zdrojů pro připojení k databázi a odesílání e-mailů v souboru *core/web/META-INF/tomcat/context.xml*:

```
<Context debug="5" reloadable="true" crossContext="true">
    <Resource name="jdbc/PnetDB" auth="Container"
        type="javax.sql.DataSource" username="pnet_user" password="pnet_user"
        driverClassName="oracle.jdbc.OracleDriver"
        url="jdbc:oracle:thin:@localhost:1521:XE"
        maxActive="8" maxIdle="4" />

    <Resource name="mail/PnetSession" auth="Container"
        type="javax.mail.Session"
        mail.smtp.host="localhost" />
</Context>
```

Výpis 10: Konfigurace serveru Tomcat – *core/web/META-INF/tomcat/context.xml*

Nakonec je třeba opravit URL adresu, na které aplikace běží, aby se správně generovaly URL adresy. V souboru *core/config/defaultsettings.properties* je to direktiva *siteHost=localhost:8090*.

2.2.2.6 Kompilace a sestavení Project.net

Jak už bylo řečeno, k sestavení aplikace Project.net se používá program Ant.

Konfigurační skript už máme nakonfigurován, takže sestavení můžeme spustit jednoduše tímto příkazem:

```
ant -buildfile=build-tomcat.xml deploy
```

Výpis 11: Sestavení aplikace pomocí programu ANT

Tím se aplikace sestaví a nahraje do Tomcatu. Je důležité, aby v tomto okamžiku byl Tomcat spuštěný – lze to zkontrolovat pomocí utility *Monitor Tomcat*, která je součástí jeho instalace.

Pokud už je aplikace jednou nainstalována na serveru, a je provedena změna zdrojových kódů, stačí spustit namísto cíle *deploy* cíl *reload*. Tím se nahrají na server změněné soubory.

Bohužel tato operace trvá o něco déle, než by bylo nezbytně nutné, protože sestavovací procedura pro Ant při každém reloadu znovu zpracovává všechny CSS a JS soubory, a provádí i další operace, které podle mého názoru nejsou při ladění programové části aplikace potřebné.

Autoři si zřejmě jsou tohoto jevu vědomi a v návodu na webu zmiňují cíle jako *redeploy-classes* a *redeploy-jsp*. Verze souboru *build-tomcat.xml*, která je součástí popisované distribuce tyto akce ovšem neobsahuje, jedná se tedy nejspíš až o funkce nové verze.

2.2.2.7 Spuštění aplikace Project.net Když už máme aplikaci zkompilovanou a nahranou na serveru, stačí do webového prohlížeče zadat adresu *http://localhost:8090/*, a dostaneme se na úvodní stránku aplikace.

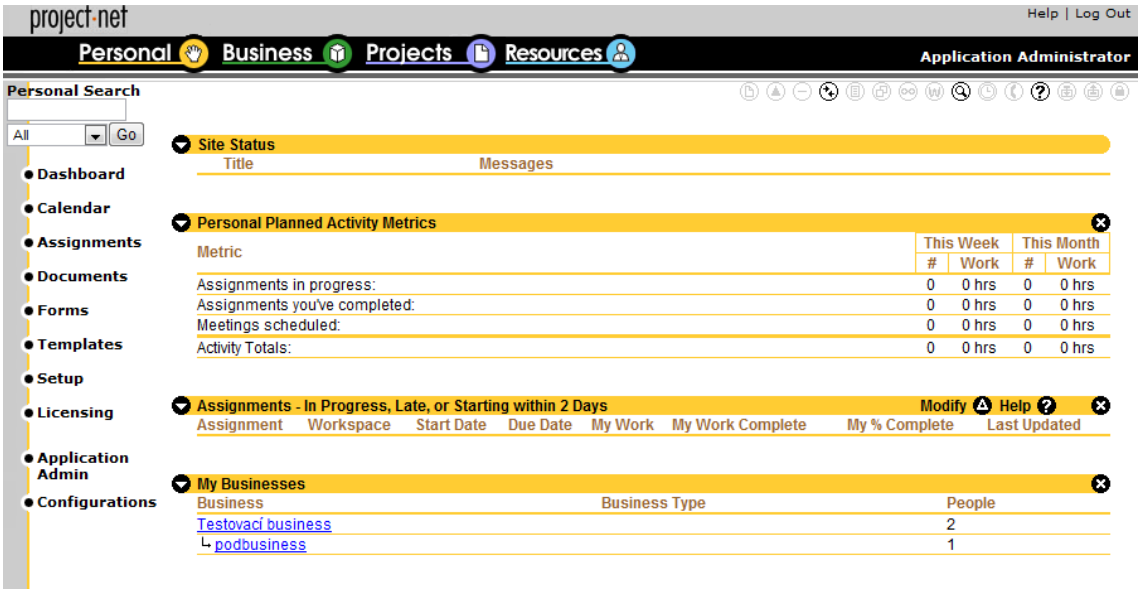
Pokud aplikace vypadá jako na obrázku 1 na stránce 6, je vše v pořádku. Obrázek ukazuje již modifikovanou verzi *v8.4_fulltext*, kromě označení verze by ale v zobrazení neměl být žádný rozdíl.

Do aplikace se přihlásíme zadáním jména `appadmin` a hesla `appadmin`.

Poslední, co je třeba pro správnou funkci managementu dokumentů udělat, je nastavení adres pro tzv. Document Vault – úložiště dokumentů. Po přihlášení jako uživatel appadmin stačí zvolit položku *Application Admin* a zde *Doc Vault*. Vložíme záznamy, označující složky, kde je možné dokumenty ukládat, např. *c:\vault\one*, *c:\vault\two* a *c:\vault\three*. Proč autoři doporučují nastavit 3 složky se mi vypátrat nepodařilo.

2.3 Popis způsobu práce s dokumenty v Project.net

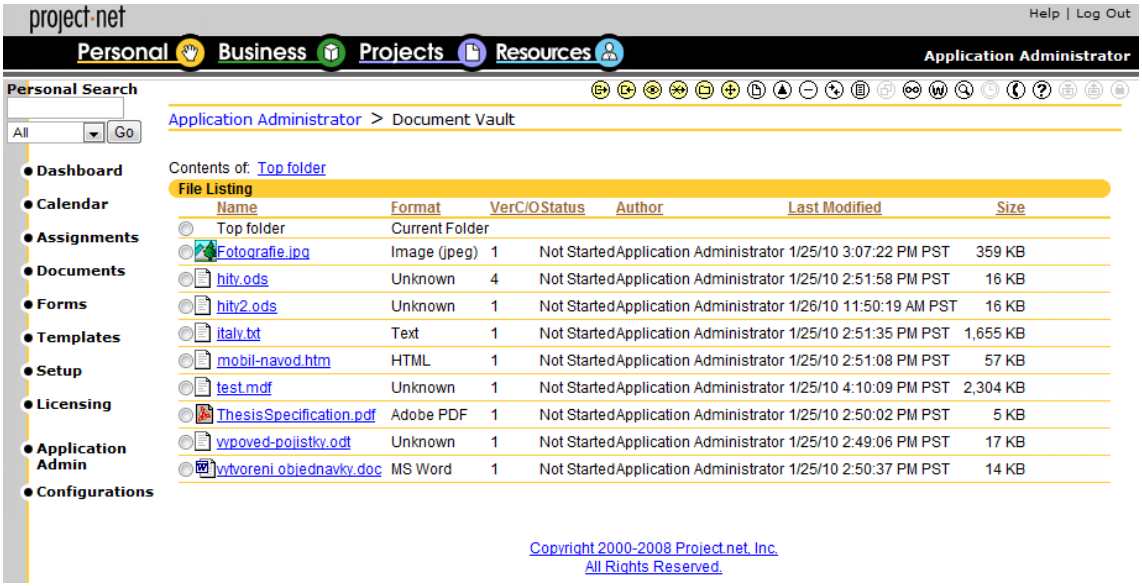
Nyní by už mělo vše pro nás podstatné fungovat. Můžeme tedy přistoupit k testování managementu dokumentů.



Obrázek 2: Hlavní stránka aplikace

Po přihlášení do aplikace se objeví stránka podobná té na obrázku 2. V horní části má uživatel na výběr z několika částí aplikace – *Personal* (osobní), *Business* (firma), *Projects* (projekty), *Resources* (zdroje). Ve výchozím zobrazení je vybráno Personal.

Dokumenty lze ukládat v sekcích Personal, Business a Projects, ve kterých je v levé části vždy položka Documents.



Obrázek 3: Seznam nahraných dokumentů

Na obrazovce s dokumenty (obrázek 3) je výpis aktuálně nahraných dokumentů, roztříděný podle adresářové struktury. Vlevo od každého názvu dokumentu je radiobutton, který umožní soubor vybrat, pokud s ním chceme provádět nějakou akci.

Akce, které lze s dokumenty provádět jsou dostupné pod ikonami vpravo nahoře na obrazovce. Jsou to hlavně *View* (zobrazení), *Import* (vložení nového dokumentu), *Create New Folder* (vytvoření složky), *View Properties* (zobrazit vlastnosti dokumentu), *Modify Properties* (nastavení vlastností dokumentu), *Remove* (odstranění dokumentu) a *Search all Documents* (vyhledávání v dokumentech).

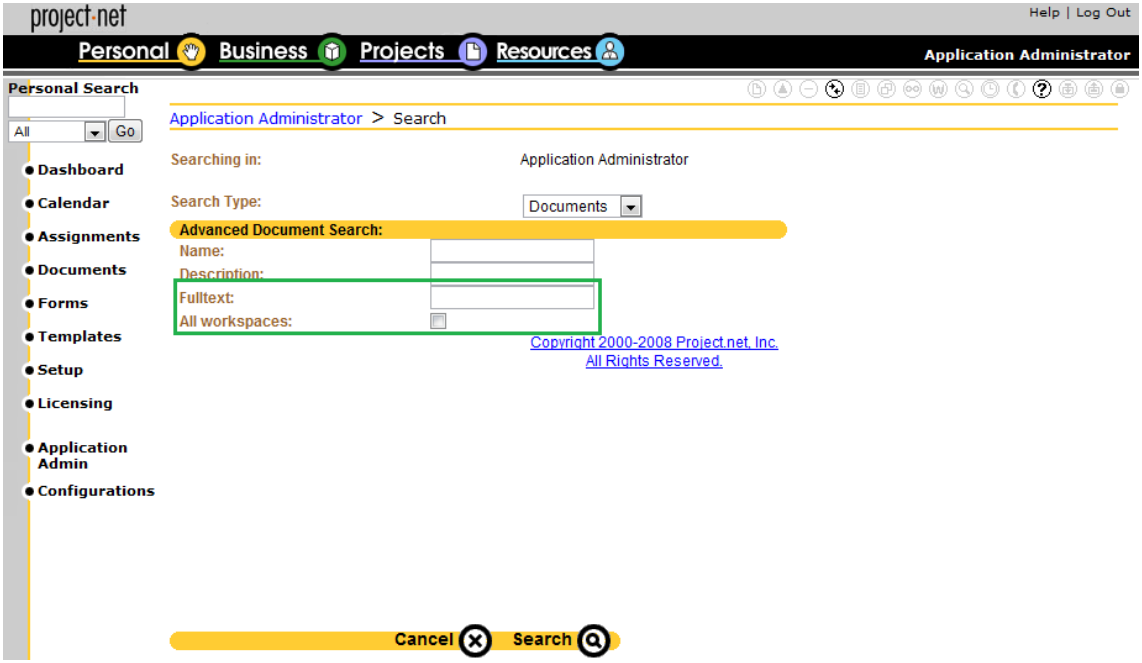
Potvrzení nebo zrušení akcí je trochu netypicky zobrazeno na samém konci stránky, tedy ne bezprostředně pod formulářem, ale úplně dole. Na větších monitorech je tedy u kratších formulářů poměrně velká mezera, mající za následek, že se volby pro odeslání nebo zrušení formuláře dají snadno přehlédnout.

K dokumentům je možné přidat kromě názvu i popisek a poznámky. Pokud je třeba nahrát dokumentů několik, je možné z nich vytvořit ZIP archiv, uploadovat jej celý, a při importu zvolit možnost *Upload files and folders from zip file*, tím se vloží všechny v archivu vložené soubory, i se zachováním adresářové struktury.

Pokud chce některý z uživatelů dokument modifikovat, zvolí *Check Out* (získat), čímž se dokument zablokuje, a nemohou ho editovat ostatní uživatelé. Po navrácení dokumentu volbou *Check In* se dokument opět odemkne, a inkrementuje se jeho číslo verze.

2.3.1 Vyhledávání v dokumentech

Vyhledávat v dokumentech je možné také z nabídky vlevo – v horní části je vždy vyhledávání v dané části (například Personal), buď to ve všech objektech, nebo je možné vybrat typ objektu, ve kterém se má vyhledávat (například dokumenty). Vyhledávání je vidět na obrázku 4.



Obrázek 4: Vyhledávání v dokumentech

Jak již bylo zmíněno v úvodu, vyhledávat lze ve výchozí verzi aplikace jen v názvech nebo popisech dokumentů. Volby *Fulltext* pro zadání fráze vyhledávané uvnitř dokumentu a *All Workspaces* pro rozšíření hledání i mimo aktuální kontext (v osobních dokumentech, dokumentech všech business i projektů), které jsou na obrázku zvýrazněny zeleně, jsou přidány až úpravami popisovanými v této práci.

2.3.2 Způsob uložení dokumentů

Ačkoli data o dokumentech, jako název, popis, verze apod. se ukládají do databáze, samotný obsah dokumentů se ukládá na disk ve formě souborů, v poněkud nepřehledné struktuře – soubory se neukládají do složek s původními názvy, ale podle ID dokumentů a nadřazených Document Space a dalších objektů z databáze. Podrobněji je o způsobu pojmenování souborů s dokumenty pojednáno v kapitole 4.5.

3 Fulltextové vyhledávání v binárních dokumentech v MSSQL 2008

Tato kapitola trochu odbočuje od aplikace Projet.net, popisuje totiž princip fulltextového vyhledávání v dokumentech tak, jak je implementováno v MSSQL 2008, tedy obecně, bez vazby na aplikaci Project.net.

Je zde popsáno, jak vytvořit nad tabulkou fulltextový index a jak formulovat vyhledávací dotazy nad takto indexovanými daty. Tyto znalosti je možné uplatnit kdekoli, nejen u aplikace Project.net.

3.1 Základy fulltextového vyhledávání

Fulltextovým vyhledáváním rozumíme prohledávání velkého množství obecného, blíže nestrukturovaného textu. Protože se jedná o množství dat, které není z časových důvodů možné při každém vyhledávání celé projít, musí se hledat jiné metody pro vyhledávání v takových datech.

Historicky první informační systémy pracující s textem to řešily omezením množství prohledávaných dat, takže uměly vyhledávat pouze v nadpisech, případně klíčových slovech či jiných heslech. Tato hesla se musela zadávat ručně během zadávání dokumentů, a obsahovala jen malou množinu toho, co mohl uživatel chtít najít.

Spolu s rozvojem těchto systémů pochopitelně vzrůstaly požadavky na vyhledávání, což pohánělo vývoj, a moderní programy proto umí mnohem více, než jen najít výskyt textového řetězce.

K těm speciálním textovým funkcím patří například nalezení slov, které se v textu vyskytují v jiném pořadí, jiném tvaru (skloňování), hledání synonym nebo třeba možnost použití logických spojek mezi hledanými výrazy.

Vylepšení vlastností, hlavně výkonu, vyhledávacích operací je dosaženo použitím tzv. fulltextových indexů. Úspěšnost těchto speciálních indexů je dána tím, že využívají podstaty textu, konkrétně jeho složení ze slov, a namísto prohledávání dat bajt za bajtem, které není efektivní, evidují slovník použitých slov a tabulku jejich výskytů v jednotlivých záznamech prohledávané tabulky.

Vzniká tedy fulltextový index, který značně zjednodušuje operaci vyhledávání tím, že namísto velkého množství textu se prohledává jen tabulka, kde je ke každému slovu uvedeno, ve kterých záznamech se nachází.

Kromě toho se používají další, spíše lingvistické techniky, jako například lemmatizace, která převádí slova do základního tvaru, aby se dala vyhledávat i slova napsaná v jiných pádech, tezaurus, sloužící pro hledání synonym a významově podobných slov, booleovská logika pro přesnější formulaci hledaného výrazu a podobně.

O tom, jak je možné tyto funkce realizovat, byla napsána spousta teorie, a není to náplní této práce, protože využívám již hotového řešení dostupného v MSSQL 2008.

Konkrétní vlastní implementaci fulltextového algoritmu popisuji například ve své bakalářské práci [11], v kapitole 4.3.

3.2 Microsoft SQL Server 2008

Americká společnost Microsoft začala vyvíjet svůj vlastní databázový server na základě programu Sybase SQL Server v roce 1989, původně pro operační systém OS/2. V roce 1993 byla Microsoftem uvolněna první verze pro platformu WinNT. Společnost Sybase nicméně ještě nějakou dobu ovlivňovala vývoj produktu, než se obě firmy rozešly [10].

Verze SQL Serveru, kterou popisuji v tomto textu má číslo 10.0 a prodává se pod obchodním označením SQL Server 2008. Je k dispozici i ve verzi Express, kterou je možné stáhnout zdarma na webu společnosti Microsoft[12].

Databáze MSSQL 2008 má spoustu zajímavých vlastností, ať už jde o pokročilé datové typy, práci s geografickými daty, správu uživatelských účtů integrovanou s účty Windows, zpracování transakcí, uložené procedury a podobně. Pro tuto práci je však nejdůležitější vlastností fulltextové vyhledávání.

3.2.1 Fulltextové vyhledávání v MSSQL 2008

3.2.1.1 Podporované formáty souborů Zatímco konkurenční produkty, zkušenosti mám například s MySQL, podporují fulltextové vyhledávání pouze nad textovými daty, MSSQL podporuje vyhledávání i v binárních dokumentech. To znamená, že v databázi je možné uložit přímo zdrojový kód dokumentu, například ve formátu PDF, a během indexace se tento dokument převede do textové podoby a obsah zaindexuje.

Server k tomu, aby dokázal získat z binárního dokumentu čistý text, potřebuje tzv. iFilter. Ve výchozí instalaci SQL Serveru není počet filtrů dostačující, chybí například soubory typu PDF, nebo formáty kancelářských balíků MS Office a OpenOffice.org.

Naštěstí lze tyto iFiltry doinstalovat[13]. Po instalaci je třeba zavolat na SQL Server proceduru: `EXEC sp_fulltext_service "update_languages"`.

Seznam aktuálně podporovaných formátů souborů lze získat zavoláním uložené procedury: `EXEC sp_help_fulltext_system_components "filter"` [12].

3.2.1.2 Vytvoření fulltextového katalogu Způsobů jak nastavit fulltextovou indexaci nad daty v MSSQL je několik. Dá se to udělat například pomocí SQL dotazů, jednodušší je ale použití aplikace SQL Server Management Studio. Jedná se o propracované GUI k databázi MSSQL, které je součástí instalace serveru.

Při spuštění nás aplikace vyzve k zadání přihlašovacích údajů, poté se zobrazí panel *Object Explorer*, kde najdeme databázi, kterou chceme fulltextově prohledávat.

Nejprve si vytvoříme fulltextový katalog – v nabídce Object Exploreru otevřeme *Storage/Full Text Catalogs* a pravým tlačítkem myši otevřeme kontextovou nápovědu, kde zvolíme *New Full-Text Catalog...*.

[12] Odkaz pro stažení: http://www.microsoft.com/express/Database/default.aspx#Installation_Options

[13] Filtry pro dokumenty PDF jsou dostupné na stránkách společnosti Adobe
http://www.adobe.com/support/downloads/detail.jsp?ftpID=2611,
pro dokumenty MS Office a OpenOffice.org je k dispozici Microsoft Office 2010 Filter Pack
http://support.microsoft.com/default.aspx?scid=kb;en-us;945934

3.2.1.2.1 Nastavení fulltextového katalogu V nově otevřeném okně zvolíme název pro fulltextový katalog, a z možností *Accent sensitivity* vybereme *Sensitive* nebo *Insensitive*.

První volba znamená, že se v hledání bude rozlišovat mezi písmeny s diakritikou a bez. Občas se totiž setkáme s dokumenty, které jsou psány bez diakritiky, a volba Accent Insensitive nám pomůže například zadáním dotazu `normální` najít i dokumenty, které obsahují slovo `normalni`.

V dnešní době se sice už nestává, že by se dokumenty psaly bez diakritiky úmyslně, jak tomu bylo v minulosti vlivem zmatků kolem kódování češtiny, ale můžeme se setkat například s tím, že texty v dokumentech exportovaných do formátu PDF z programu LaTeX nebo jiných jsou psány bez diakritiky, a diakritická znaménka jsou k písmenkům „dotisknuta“ zvlášť.

Proto doporučuji zvolit Accent Insensitive, aby se předešlo tomu, že vyhledávač nenajde dokument při zadání slova s diakritikou, a při hledání stejného výrazu bez diakritiky ano, přestože v dokumentu vypadá vše správně s diakritikou.

Drobnou nevýhodou tohoto přístupu je, že může nastat situace, kdy hledáme určité slovo, a ve výsledcích se objeví dokumenty obsahující jiné slovo, lišící se v diakritických znaménkách. Například při hledání `kos` (pták) vyhledávač najde dokumenty obsahující slovo `koš` (kontejner). Z vlastní zkušenosti ale vím, že je lepší, když vyhledávač občas vrátí výsledků více, ve kterých si uživatel vybere, než když ve výsledcích není to, co uživatel hledá.

Volbu Accent Sensitive je tedy vhodné použít v případě, že jsme si jisti, že všechny dokumenty jsou korektně napsány s diakritickými znaménky, a víme, že uživatelé při zadávání hledaných dotazů budou diakritiku používat rovněž.

3.2.1.3 Vytvoření fulltextového indexu Když už máme vytvořen fulltextový katalog, je načase do něj přidat nějaký fulltextový index. Klikneme tedy pravým tlačítkem v Object Exploreru na tabulku, nad níž chceme index vytvořit a z kontextového menu vybereme volbu *Full-Text index / Define Full-Text Index...*.

3.2.1.3.1 Výběr polí pro indexaci, nastavení dělení slov a popisu typu dokumentů Otevře se nám průvodce, ve kterém vybereme *Unique index* – klíč, který bude použit k jednoznačné identifikaci záznamů v tabulce.

V dalším kroku vybereme pole, která chceme indexovat. Pro naše účely by to mělo být pole s datovým typem *varbinary(max)*, ale použít lze libovolné textové pole nebo obrázek.

Dále pro každý vybraný řádek můžeme zvolit *Language for Word Breaking*. Jedná se o pravidla, podle kterých bude prováděno rozdělení textu na slova. K dispozici jsou pravidla pro různé jazyky, čeština mezi nimi bohužel chybí.

Nicméně dělení slov je podobné ve většině jazyků, takže je možné použít i dělení slov typu *Neutral*, které rozděluje slova podle neutrálních znaků, tedy mezer a interpunkčních znamének.

Poslední vlastnost – *Type Column* – označuje pole tabulky, ve kterém je uložena informace o typu dokumentu uloženém v indexovaném poli. Je-li tedy ve sloupci s dokumentem uložen binární dokument typu PDF, je v sloupci označující typ uložen řetězec `.pdf`.

Podle tohoto sloupce pozná fulltextový stroj, jaký filtr má na binární data použít. Pokud není vlastnost Type Column vyplněna, předpokládá se, že není v datovém sloupci uložen binární dokument, ale prostý text.

Pro jeden index můžeme zkombinovat více sloupců, a je tedy možné vyhledávat například v názvu, popisu i obsahu dokumentu najednou.

3.2.1.3.2 Nastavení generování indexů Když máme hotovo nastavení, co se bude indexovat, přistoupíme do dalšího kroku, kde si vybereme, jak bude index generován. Volba *Automatically* znamená, že se o aktualizaci fulltextového indexu bude starat server automaticky, volba *Manually* znamená, že se indexace bude provádět pouze na explicitní požadavek. Volba *Do not track changes* umožňuje zrušit provedení okamžité indexace po dokončení průvodce. Osobně doporučuji zvolit automatické indexování.

3.2.1.3.3 Výběr fulltextového katalogu a stoplistu V dalším kroku vybereme fulltextový katalog, do nějž bude vytvářený index umístěn. Případně můžeme vytvořit pro daný index nový katalog.

Dále je zde možné vybrat *full-text stoplist* – seznam slov, která jsou vyloučena z indexování. Jedná se nejčastěji o slova, která se v textech vyskytují příliš často, a nemají relevantní význam. Takovými slovy jsou například spojky, zájmena apod.

Nový stoplist je možné vytvořit přes Object Explorer v nabídce *Storage / Full Text Stoplists* volbou *New Full-Text Stoplist...*. Výchozí systémový stoplist, který je v SQL Serveru dostupný po instalaci obsahuje seznam celkem téměř 15 000 stop slov pro více jak 40 jazyků, čeština mezi nimi bohužel chybí. Předpokládám ale, že to dá Microsoft brzy do pořádku, a v další verzi SQL Serveru již podpora pro češtinu bude.

Kompletní seznam systémovým stoplistem podporovaných jazyků s počtem stop slov lze ze serveru získat tímto dotazem:

```
1 SELECT L.name AS [lanugage name], count(W.stopword) AS [stopwords count]
2 FROM sys.fulltext_languages L JOIN sys.fulltext_system_stopwords W ON W.language_id = L.lcid
3 GROUP BY L.name
4 WITH CUBE
```

3.2.1.3.4 Dokončení vytváření indexu V dalším kroku je volitelně možné upravit časový plán indexace dat. Poslední krok průvodce sumarizuje všechny zadané údaje a tlačítko *Finish* spustí všechny potřebné akce a SQL dotazy. Nyní je databáze připravena pro fulltextové vyhledávání.

3.2.1.4 Dotazy pro fulltextové vyhledávání Nad vytvořenými fulltextovými indexy lze provádět dva typy dotazů [13]:

- **fulltextové predikáty** – vrací ano/ne, jestli se slovo v textu vyskytuje
- **fulltextové funkce** – vrací tabulku výskytů i s relevancí

Predikát `CONTAINS ( column_list , search_condition )` lze použít uvnitř klauzule `WHERE` pro omezení vyhledaných záznamů podle fulltextové podmínky, například tato:

```
1 SELECT title, description FROM documents WHERE CONTAINS(data, 'diplomova prace')
```

Funkce `CONTAINSTABLE ( table_name , column_list , search_condition )` naproti tomu vrací tabulku, která se skládá ze dvou sloupců – `KEY` je klíč řádku, který obsahuje shodu s hledaným dotazem a `RANK` je číselné ohodnocení relevance daného záznamu k uvedenému dotazu. Hodnoty jsou celá čísla od 1 do 1000, kde 1000 je nejlepší shoda.

Tuto tabulku lze pomocí klauzule `JOIN` spojit s vyhledávanými daty a seřadit podle relevance, například takto:

```
1 SELECT title, description
2 FROM documents D
3 JOIN CONTAINSTABLE(documents, data, 'diplomova prace') C
4     ON C.KEY = D.id
5 ORDER BY C.RANK DESC
```

Jako parametr *search_condition* je možné použít jedno hledané slovo nebo několik slov, ale i složitější výrazy, jako například:

- **logické spojky** – `diplomova OR prace`
- **hledání prefixu** – `diplom*`
- **výskyt dvou slov blízko sebe** – `diplomova NEAR prace`
- **využití thesauru** a další...

Kromě toho existuje ještě predikát `FREETEXT` a funkce `FREETEXTTABLE`, které mají stejný účel i způsob použití, liší se pouze v tom, že zatímco `CONTAINS`... interpretují mezi slovy uvedené operátory, `FREETEXT`... interpretují vše jako slova, a proto nelze v jejich argumentu používat žádné logické spojky a podobně.

3.3 Srovnání fulltextového vyhledávání s LIKE

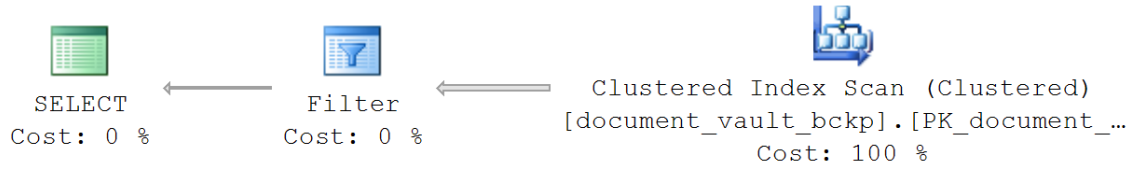
Vyhledávání podle řetězce v textových datech je samozřejmě možné realizovat i pomocí operátoru `LIKE`, například takto:

```
1 SELECT title, description FROM documents WHERE data LIKE '\%diplomova prace\%'
```

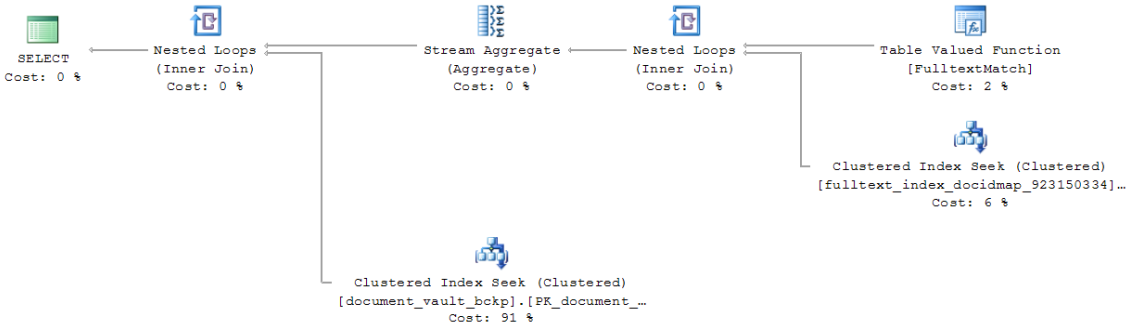
Použití vypadá o mnoho jednodušeji, navíc nevyžaduje vytváření fulltextových katalogů, generování fulltextových indexů a další operace náročné na čas i paměťový prostor.

SQL Server Management studio umožňuje zobrazit pro každý dotaz tzv. *Execution plan*, který znázorňuje, jak probíhá zpracování dotazu. Zobrazí dotaz jako vzájemnou interakci jednodušších funkcí. Je to užitečný nástroj pro ladění složitějších dotazů.

Pro zajímavost uvádím na obrázcích 5 a 6 execution plan dotazu pro vyhledání dokumentů obsahujících slovo „test“ jak pomocí `LIKE`, tak pomocí fulltextu. Je jasně vidět, že fulltextové hledání je složitější.



Obrázek 5: Execution plan vyhledávání pomocí LIKE



Obrázek 6: Execution plan vyhledávání pomocí fulltextu

Nabízí se otázka, proč vůbec vymýšlet nějaké fulltextové hledání, když existuje jednodušší řešení. Bohužel ne vždy je nejjednodušší řešení to nejlepší. Vyhledávání pomocí klauzule `LIKE` má hned několik nevýhod:

- **vyhledává pouze v textových datech** – Neumožňuje prohledávat obsah binárních dokumentů. Pokud bychom chtěli realizovat prohledávání binárních dokumentů, museli bychom napsat filtry pro jednotlivé typy souborů a do databáze ukládat kromě binárních dat i čistě textový obsah.
- **neumožňuje formulovat složitější dotazy** – Klauzule `LIKE` interpretuje text jako posloupnost znaků, neumí pracovat se slovy a souvislostmi mezi nimi, neexistuje nic jako thesaurus nebo logické spojky. Pokud bychom chtěli něco podobného realizovat, potřebovali bychom další aplikační logiku, která by formulovala složitější dotazy. Zpracování takových dotazů by navíc trvalo mnohem déle.
- **neumožňuje hodnotit relevanci** – Pomocí LIKE sice zjistíme, jestli se dané slovo v textu vyskytuje nebo ne, nejsme ale schopni zjistit, jestli se vyskytuje vícekrát, na začátku nebo na konci dokumentu a podobně.
- **je pomalé** – Ačkoli je vyhledávání pomoci LIKE algoritmicky jednodušší, je časově mnohonásobně složitější, než hledání pomocí fulltextu.

3.3.1 Výkonnostní srovnání

U rychlosti obou metod hledání ještě chvíli zůstanu, protože jsem se rozhodl porovnat rychlost nalezení výsledků (vyhledání jednoho slova) pomocí obou metod.

K tomu jsem si vytvořil testovací databáze s dokumenty o velikosti přibližně 15 MB, 30 MB, 60 MB, 120 MB, resp. 240 MB, a na každé z nich spustil obě metody vyhledávání. Abych omezil vliv náhody, provedl jsem každé měření $5\times$, nicméně rozdíl v časech je tak markantní, že to ani nebylo třeba. Výsledky měření ukazuje tabulka 3, průměrné hodnoty jsou vyneseny do grafu na obrázku 7.

Problém LIKE hledání není ani tak v tom, že dlouho trvá, ale spíše v tom, že doba hledání roste přímo úměrně s velikostí databáze. Je to proto, že LIKE musí vždy přečíst celý obsah databáze.

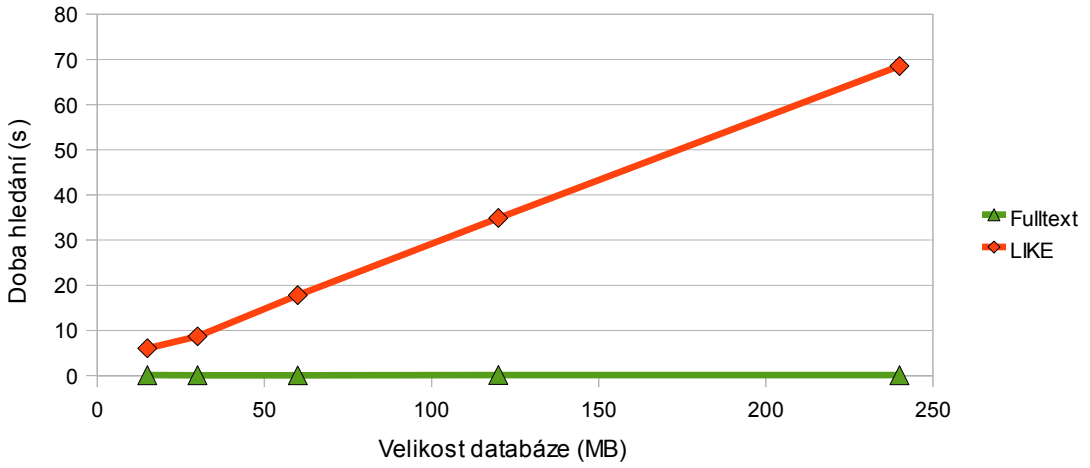
Naproti tomu fulltextové hledání prohledává jen index, a proto je méně závislé na velikosti databáze. To, co rychlost hledání omezuje, je velikost indexu, která je závislá spíše na tom, kolik unikátních slov se ve vyhledávaném textu vyskytuje, tedy na velikosti slovníku, a na tom, jaká je frekvence výskytu těchto slov.

Přesně vyjádřit časovou složitost fulltextového vyhledávání je sice složité, výsledek experimentu je ovšem jasný. Zatímco hledání pomocí LIKE je únosné pro databáze velikosti jednotek megabajtů, fulltextové hledání umožňuje efektivně prohledávat databáze o velikosti gigabajtů.

O možnostech MSSQL Serveru, stejně jako o fulltextovém vyhledávání se toho dá napsat ještě více, ale pro účely této práce stačí to, co bylo zmíněno v této kapitole, a nemá příliš cenu zacházet do ještě větších podrobností.

velikost databáze (MB)	čas fulltext (ms)	čas LIKE (ms)
15	117	6 066
	121	6 070
	127	6 102
	119	6 018
	117	6 028
průměr	**120**	**6 057**
30	81	8 700
	77	8 602
	122	8 936
	130	8 648
	84	8 631
průměr	**99**	**8 703**
60	98	17 744
	80	17 643
	66	18 021
	29	17 925
	95	17 688
průměr	**74**	**17 804**
120	84	34 580
	76	35 252
	377	34 695
	102	35 223
	78	34 687
průměr	**143**	**34 887**
240	175	70 778
	147	68 384
	76	67 808
	83	67 813
	98	67 851
průměr	**116**	**68 527**

Tabulka 3: Výsledky měření doby potřebné pro nalezení dokumentů



Obrázek 7: Srovnání rychlosti hledání pomocí fulltextu a klauzule LIKE

4 Propojení Project.net a MSSQL 2008, realizace fulltextového vyhledávání v dokumentech

V této kapitole popíšu konkrétní úpravy aplikace Project.net, které jsem provedl během implementace vyhledávání.

První část popisuje požadavky kladené na vyhledávání a propojení databází. Další část popisuje návrh způsobu propojení obou databází a úprav funkcí Project.net. Dále jsou v textu popsány konkrétní poznatky, spolu s ukázkami zdrojových kódů z reálné implementace. V závěru kapitoly je popsán program pro usnadnění nasazení hotového řešení.

4.1 Požadavky na propojení

S vedoucím práce jsme definovali následující klíčové požadavky na výsledné řešení:

- **Indexace binárních formátů** – Požadavkem je, aby systém dokázal vyhledávat v nejběžnějších formátech dokumentů, kromě textových souborů typu TXT, HTML, XML jsou to také binární formáty PDF, DOC, DOCX a ODT.
- **Jednoduché nastavení a obsluha** – Z uživatelského hlediska musí být fulltextové vyhledávání stejně jednoduché, jako standardní hledání v aplikaci Project.net.

  Konfigurace má být jednorázová během nasazení a dále už by neměla být nutná žádná údržba.
- **Možnost hledání napříč aplikací** – Uživatel má mít možnost vyhledávat nejen v aktuálně otevřené agendě, ale i najednou v osobních dokumentech a dokumentech všech businessů a projektů. Ve výsledcích hledání by mělo být jasně patrné, do které agendy dokumenty patří.
- **Řazení výsledků dle relevance** – Nalezených dokumentů může být hodně, je tedy logický požadavek, aby byly na začátku výsledků ty dokumenty, které odpovídají hledanému řetězci nejvíce.
- **Respektování uživatelských práv** – Project.net sice nezobrazí uživateli dokument, na který nemá práva pro čtení, nicméně by se tyto dokumenty neměly ve výsledcích vyhledávání ani objevovat. Jednak to zpřehlední práci, a jednak to zabrání tomu, aby se uživatel dověděl o existenci dokumentů, o nichž nemá vědět.
- **Minimální zásahy do existujících souborů** – Vzhledem k tomu, že je možné, že bude v budoucnu třeba upgradovat verzi aplikace Project.net, je třeba provádět úpravy tak, aby došlo k minimálním změnám zdrojového kódu. Maximum funkcí je třeba soustředit do nových souborů. Tím se značně ulehčí případné aktualizace zdrojových kódů.

O tom, jak a do jaké míry je každý z požadavků splněn, je pojednáno v závěru.

4.2 Návrh datových struktur pro uložení obsahu dokumentů

V databázi MSSQL je třeba uchovávat data dokumentů – obsah binárních souborů. Při fulltextovém vyhledávání chceme prohledávat i názvy a popisky souborů, proto je lepší je v databázi uchovávat taky.

Kromě toho potřebujeme mít v databázi jednoznačnou identifikaci dokumentu, to proto, abychom dokázali spárovat záznam ve fulltextové databázi s dokumentem.

S ohledem na tyto skutečnosti jsem navrhl strukturu tabulky *pnet_document_vault*, která slouží jako úložiště pro dokumenty z aplikace Project.net. Její struktura je popsána tabulkou 4.

název sloupce	datový typ	null	popis
document_id	numeric(20,0)	NOT NULL	id dokumentu v Project.net
name	nvarchar(255)	NOT NULL	název dokumentu
description	nvarchar(4000)	NOT NULL	popis dokumentu
data	varbinary(max)	NOT NULL	data dokumentu (obsah souboru)
type	nvarchar(10)	NOT NULL	typ souboru (přípona s tečkou)

Tabulka 4: Struktura tabulky pnet_document_vault

Fulltextový index jsem pak vytvořil nad poli *data*, *name* a *description*, s dělením slov *Neutral*. Pro sloupec *data* je *Type Column* sloupec *type*, ostatní jsou textová data, takže mají *Type Column* prázdný.

4.3 Realizace propojení Project.net s databází MSSQL 2008

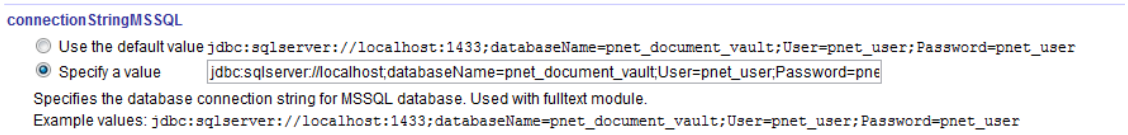
Aby mohla aplikace Project.net, psaná v jazyce Java pracovat s daty uloženými v MSSQL databázi, je třeba ovladač databáze.

V Javě se používá rozhraní JDBC[14], což je jednotné rozhraní pro práci s relačními databázemi. Pro připojení ke konkrétnímu databázovému serveru je však potřeba ještě speciální JDBC ovladač pro danou databázi. JDBC ovladač pro MSSQL je k dispozici ke stažení přímo na stránkách společnosti Microsoft[15].

Pro připojení k databázi prostřednictvím JDBC je třeba zadat URL adresu SQL Serveru. Tato adresa vypadá například takto:

`jdbc:sqlserver://`*(IP adresa nebo doménové jméno serveru)*`:`*(číslo portu)*`;`
`databaseName=`*(jméno databáze)*`;User=`*(jméno uživatele)*`;Password=`*(heslo)*

Proto, aby šla tato URL nastavovat co nejpohodlněji jsem ji přidal mezi nastavení v menu *Personal / Application Admin / Manage / Sys. Settings*. Vzhled nastavení lze vidět na obrázku 8.



Obrázek 8: Ukázka nastavení URL databáze v administraci aplikace

Ve skutečnosti k přidání tohoto nastavení do konfigurace stačilo připsat do konfiguračního souboru *defaultsettings.properties* tyto řádky:

```
206 connectionStringMSSQL=jdbc:sqlserver://localhost:1433;databaseName=pnet_document_vault;User
        =pnet_user;Password=pnet_user
207 connectionStringMSSQL.seq=30
208 connectionStringMSSQL.example.1=jdbc:sqlserver://localhost:1433;databaseName=
        pnet_document_vault;User=pnet_user;Password=pnet_user
209 connectionStringMSSQL.description=\
210 Specifies the database connection string for MSSQL database. Used with fulltext module.
```

Výpis 12: Soubor core/config/defaultsettings.properties

O zbytek, tedy hlavně možnost změny nastavení z administrace, se aplikace Project.net postará sama.

[14] http://java.sun.com/javase/technologies/database/

[15] http://www.microsoft.com/downloads/en/confirmation.aspx?familyId=99b21b65-e98f-4a61-b811-19912601fdc9&displayLang=en

4.3.1 Reakce aplikace na přidání, změnu a smazání dokumentu

Do aplikace Project.net je potřeba vložit kód na místa, kde se nějakým způsobem manipuluje s dokumenty. Abych co nejméně modifikoval stávající soubory, vytvořil jsem novou třídu *DocumentFulltext* v balíku *net.project.util*. Do této třídy jsem se snažil dostat maximum kódu, v podobě metod, a z ostatních částí aplikace je pouze volat.

4.3.1.1 Připojení k MSSQL databázi Třída *DocumentFulltext* obsahuje privátní vlastnost *ConnectionString*, která uchovává URL pro připojení k databázi. Tato hodnota se získá při inicializaci z nastavení aplikace:

```
27 private static final String ConnectionString = Compatibility.getConfigurationProvider().getSetting
       ("connectionStringMSSQL");
```

Výpis 13: Vlastnost *DocumentFulltext.ConnectionString*

Pro získání objektu pracujícího s databází slouží metoda *getConnection()*:

```
37 private static Connection getConnection() throws SQLException {
38     try {
39         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
40
41     } catch (ClassNotFoundException e) {
42         Logger.getLogger(FileManager.class).error("Error getting database driver: " + e);
43         throw new SQLException("Error getting database driver: " + e, e);
44     }
45
46     return java.sql.DriverManager.getConnection(ConnectionString);
47 }
```

Výpis 14: Metoda *DocumentFulltext.getConnection()*

4.3.1.2 Vložení nového dokumentu Pokud uživatel nahraje do aplikace nový dokument, zpracovává se upload souboru v metodě *upload()* třídy *FileManager* z balíku *net.project.document*. Do ní jsem tedy přidal volání metody *saveFileToDatabase()*:

```
57 public static void saveFileToDatabase(String documentID, File file, String fileType) throws
       FileUploadStorageException {
58     Connection connection = null;
59     PreparedStatement statement = null;
60
61     try {
62         connection = getConnection();
63
64         statement = connection.prepareStatement("DELETE FROM [document_vault] WHERE [
               document_id] = ?");
65         statement.setString(1, documentID);
66         statement.execute();
67         statement.close();
68
69         FileInputStream fileInputStream = new FileInputStream(file);
70         statement = connection.prepareStatement("INSERT INTO [document_vault] ([document_id], [
               name], [description], [data], [type]) VALUES (?, '', '', ?, ?)");
71         statement.setString(1, documentID);
72         statement.setBinaryStream(2, fileInputStream, (int) file.length());
73         statement.setString(3, "." + fileType);
74         statement.executeUpdate();
75         fileInputStream.close();
76         statement.close();
77
78         connection.close();
79
80     } catch (SQLException e) {
81         Logger.getLogger(FileManager.class).error("Error creating entry in fulltext database after
               upload: " + e);
82         throw new FileUploadStorageException("Error creating entry in fulltext database after upload
               . Admin: make sure you have setup the database for fulltext.: " + e, e);
83
84     } catch (IOException e) {
85         Logger.getLogger(FileManager.class).error("Error copying uploaded file to fulltext database
               : " + e);
86         throw new FileUploadStorageException("Error copying uploaded file to fulltext database: " +
               e, e);
87     }
88 }
```

Výpis 15: Metoda *DocumentFulltext.saveFileToDatabase()*

Tato metoda napřed zkusí vymazat záznam s daným ID dokumentu, pokud už v databázi existuje, a pak do databáze vloží nový záznam. Pole *data* datového typu *varbinary(max)* obsahuje celý kód dokumentu, de facto kopii souboru.

Takto velké množství dat není možné načíst celé najednou do paměti, a pak předat jako parametr SQL dotazu. Proto se hodí možnost JDBC předávat BLOB data jako datový proud (stream). Stačí soubor otevřít, proud předat jako parametr metodě *setBinaryStream()*, a SQL Server si pak data postupně načte.

4.3.1.3 Kopírování dokumentu Kopírování souboru se děje v metodě *copyFileForDocument()* třídy *FileManager()*. Do ní jsem přidal volání metody *copyFileInDatabase()*:

```
97 public static void copyFileInDatabase(String sourceDocumentID, String targetDocumentID)
       throws FileCopyException {
98     Connection connection;
99     PreparedStatement statement;
100
101    try {
102        connection = getConnection();
103        statement = connection.prepareStatement("INSERT INTO [document_vault] ([document_id], [
               name], [description], [data], [type]) SELECT ?, [name], [description], [data], [type]
               FROM [document_vault] WHERE [document_id] = ?");
104        statement.setString(1, targetDocumentID);
105        statement.setString(2, sourceDocumentID);
106        statement.execute();
107        statement.close();
108        connection.close();
109
110    } catch (SQLException e) {
111            Logger.getLogger(FileManager.class).debug("Error copying fulltext database entry: " + e, e
                   );
112            throw new FileCopyException("Error copying fulltext database entry: " + e, e);
113    }
114 }
```

Výpis 16: Metoda *DocumentFulltext.copyFileInDatabase()*

V této metodě je známo jen ID dokumentu, ze kterého se mají zkopírovat data, a ID dokumentu do kterého se kopírují. Díky dotazu typu `INSERT INTO ... SELECT ...` je možné vytvořit nový záznam jako kopii staršího, jen se změnou ID dokumentu. Není tedy nutné načítat žádné soubory z disku, a vše se děje jen na straně databáze.

4.3.1.4 Změna názvu nebo popisu souboru V aplikaci Project.net se při uploadu dokumentu zadává i jeho název a popis. Později možné tyto hodnoty aktualizovat.

Třída *Document* má settery *setName()* a *setDescription()*, které se volají při změně i při vkládání. Do nich jsem přidal volání svých metod *setFileName()* a *setFileDescription()*, jejichž funkce je obyčejný `UPDATE`.

4.3.1.5 Mazání dokumentu Mazání dokumentů realizuje v aplikaci Project.net metoda *remove()* třídy *Document*, do níž jsem přidal volání své metody *DeleteFileFromDatabase()*, která provádí vymazání záznamu z fulltextové tabulky, podle zadaného ID dokumentu.

4.3.1.6 Nadbytečnost Document Vaultu Klasický souborový Document Vault, by bylo možné plně nahradit ukládáním souborů v databázi. S vedoucím práce jsme se ale dohodli, že těch několik ušetřených MB na disku nestojí za to, aby se dalšími úpravami aplikace riskovalo, že by se tím zkomplikovaly updaty a zvýšilo riziko chyby.

4.3.2 Úpravy funkce vyhledávání v Project.net

Teď už jsme ve stavu, kdy aplikace replikuje úpravy dokumentů do fulltextové databáze. Stačí tedy už jen upravit vyhledávání tak, aby tato data používalo.

Vyhledávat lze v aplikaci Project.net v různých typech objektů. Základní vyhledávací funkce řeší abstraktní třída *GenericSearch* z balíku *net.project.search*, z níž vždy dědí třídy určené pro vyhledávání v určitém typu objektu. Vyhledávání v dokumentech řeší třída *DocumentSearch*, kterou jsem upravoval.

Třída *DocumentSearch* řeší několik typů vyhledávání – *Search*, *KeywordSearch*, *SimpleSearch* a *AdvancedSearch*. Fulltextové vyhledávání jsem přidával pouze do posledně zmiňovaného.

4.3.2.1 Vyhledávací formulář Napřed jsem doplnil metodu *getAdvancedSearchForm()*, která vrací HTML kód vyhledávacího formuláře. Do něj jsem doplnil pole pro zadání dotazu pro fulltextové hledání a check-box pro volbu, že se má vyhledávat ve všech agendách:

```
147 formString.append("\n<tr><td> </td>");
148 formString.append("\n<TD class=\"tableHeader\">" + PropertyProvider.get("prm.global.search.
        document.advanced.fulltext.label") + "  </TD>"); // Fulltext:
149 formString.append("\n<TD class=\"tableContent\">");
150 formString.append("<INPUT TYPE=\"TEXT\" NAME=\"FULLTEXT\">");
151 formString.append("</TD><td> </td></TR>");
152 formString.append("\n<tr><td> </td>");
153 formString.append("\n<TD class=\"tableHeader\">" + PropertyProvider.get("prm.global.search.
        document.advanced.all_workspaces.label") + "  </TD>"); // All workspaces:
154 formString.append("\n<TD class=\"tableContent\">");
155 formString.append("<INPUT TYPE=\"CHECKBOX\" NAME=\"ALL_WORKSPACES\" VALUE=\"
        YES\">");
156 formString.append("</TD><td> </td></TR>");
```

Výpis 17: Metoda *DocumentSearch. getAdvancedSearchForm()*

4.3.2.2 Hledání Vlastní hledání provádí metoda *doAdvancedSearch()* (znakem + jsou uvozeny přidané nebo změněné řádky):

```
322 public void doAdvancedSearch(HttpServletRequest m_request) {
323     StringBuffer query = new StringBuffer();
324     String name;
325     String desc;
326 +   String fulltext ;
327
328
329     m_results = new ArrayList();
330     name = m_request.getParameter("NAME");
331     desc = m_request.getParameter("DESC");
332 +    fulltext = "" + m_request.getParameter("FULLTEXT");
333 +   allWorkspaces = m_request.getParameter("ALL_WORKSPACES") != null;
334
335 +   query.append("SELECT * FROM (");
```

```
336 + query.append("SELECT doc.doc_id as id, doc.doc_name as name, doc.doc_description as
        description, '" + ObjectType.DOCUMENT + "' as object_type, pn_space_view.space_type as
        space_type, pn_space_view.space_name as space_name ");
337 + if ( fulltext .length() > 0) {
338 +     query.append(", ft.rank ");
339 + } else {
340 +     query.append(", 0 as rank ");
341 + }
342 + query.append(" FROM pn_doc_by_space_view doc, pn_space_has_doc_space, pn_space_view ");
343 + if ( fulltext .length() > 0) {
344 +     query.append(" , tmp_fulltexttable ft ");
345 + }
346 + if (allWorkspaces) {
347 +     query.append(" WHERE pn_space_view.space_id IN (SELECT space_id FROM
        pn_space_has_person WHERE person_id = " + SessionManager.getUser().getID() + ") ");
348 + } else {
349 +     query.append(" WHERE pn_space_has_doc_space.space_id = ");
350 +     query.append(getFirstSpaceID() + " ");
351 + }
352 + query.append(" AND pn_space_has_doc_space.doc_space_id = doc.doc_space_id and doc.
        is_hidden='0' AND pn_space_has_doc_space.space_id = pn_space_view.space_id ");
353
354     if (name != null && name.length() > 0) {
355         query.append("AND UPPER(doc.doc_name) LIKE UPPER('%");
356         query.append(name);
357         query.append("%') ");
358     }
359     if (desc != null && desc.length() > 0) {
360         query.append(" AND UPPER(doc.doc_description) LIKE UPPER('%");
361         query.append(desc);
362         query.append("%') ");
363     }
364 +   if ( fulltext .length() > 0) {
365 +       query.append(" AND doc.doc_id = ft.key ");
366 +   }
367
368     query.append(" AND doc.record_status = 'A'");
369
370
371 + query.append(" UNION select b.bookmark_id as id, b.name, b.description, '" + ObjectType.
        BOOKMARK + "' as object_type, null as space_type, null as space_name, 0 as rank ");
372     query.append(" from pn_bookmark_view b where b.owner_space_id = " + getFirstSpaceID());
373
374     if (name != null && name.length() > 0) {
375         query.append(" AND UPPER(b.name) LIKE UPPER('%");
376         query.append(name);
377         query.append("%') ");
378     }
379     if (desc != null && desc.length() > 0) {
380         query.append(" AND UPPER(b.description) LIKE UPPER('%");
381         query.append(desc);
382         query.append("%') ");
383     }
384
```

```
385     query.append("AND b.record_status = 'A'");
386 +   query.append(") ORDER BY rank DESC");
387
388     try {
389 +       m_db.setAutoCommit(false);
390 +           if ( fulltext .length() > 0) {
391 +               DocumentFulltext.getFulltextTable( fulltext , m_db);
392 +           }
393         m_db.executeQuery(query.toString());
394         while (m_db.result.next()) {
395             String [] result_row = {m_db.result.getString("id"),
396                 m_db.result.getString("name"),
397                 m_db.result.getString("description"),
398                 m_db.result.getString("object_type"),
399 +               m_db.result.getString("space_type"),
400 +               m_db.result.getString("space_name")};
401             m_results.add(result_row);
402         }
403 +       m_db.commit();
404 +       m_db.setAutoCommit(true);
405         m_db.release();
406     } catch (SQLException sqle) {
407         Logger.getLogger(DocumentSearch.class).error("DocumentSearch.doSimpleSearch() threw
                an SQL exception");
408     } finally {
409         m_db.release();
410     }
411
412     return;
413 }
```

Výpis 18: Metoda *DocumentSearch.doAdvancedSearch()*

Tato metoda poskládá SQL dotaz pro vyhledávání, provede jej a výsledky uloží do proměnné *m_results*. Moje úprava spočívá v tom, že se před vlastním hledáním zavolá metodu *getFulltextTable()*. Ta do připravené dočasné tabulky *tmp_fulltexttable* zkopíruje výsledek fulltextového hledání získaného z databáze MSSQL. Tato dočasná tabulka se pak použije ve vyhledávacím dotazu.

Dočasná (global temporary) tabulka v databázi ORACLE funguje tak, že samotná struktura tabulky je globální – sdílí ji všechny relace a transakce, zatímco data uvnitř jsou platná pouze pro konkrétní transakci. To je důvod, proč je vyhledávání uzavřeno do transakce. Ostatní transakce tato data nevidí, takže se nemůže stát, že by se uživatelům zamíchaly výsledky vyhledávání.

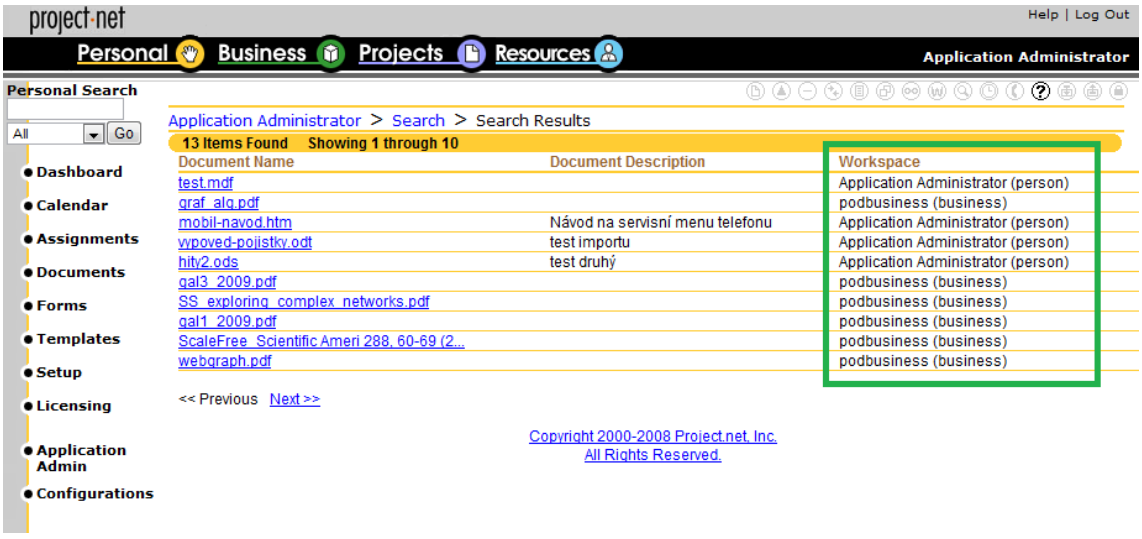
4.3.2.3 Zobrazení výsledků Výsledky vyhledávání potřebujeme zobrazit uživateli. K tomu slouží metoda *getXMLResults()* třídy *DocumentSearch*, která výsledky uložené v proměnné *m_results* naformátuje jako XML dokument, který se dále zpracovává pro zobrazení. Já jsem do této metody přidal kód pro zobrazení ze které agendy je dokument, pokud se hledá ve všech agendách:

```
523 public String getXMLResults(int start, int end) {
524     if (m_results.size() < 1)
525         return null;
526 
527     int m_start = start;
528     int m_end = end;
529 
530     if (m_start > m_results.size())
531         m_start = m_results.size();
532 
533     if (m_start < 1)
534         m_start = 1;
535 
536     if (m_end > m_results.size())
537         m_end = m_results.size();
538 
539     if (m_end < 1)
540         m_end = 1;
541 
542     m_start--;
543     m_end--;
544 
545 
546     StringBuffer sb = new StringBuffer();
547     sb.append("<channel>\n");
548     sb.append("<table_def>\n");
549     sb.append("<col>" + PropertyProvider.get("prm.global.search.document.results.name.column")
            + "</col>\n"); // Document Name
550     sb.append("<col>" + PropertyProvider.get("prm.global.search.document.results.description.
            column") + "</col>\n"); // Document Description
551 +   if (allWorkspaces) {
552 +       sb.append("<col>" + PropertyProvider.get("prm.global.search.document.results.space.
          column") + "</col>\n"); // Workspace
553 +   }
554     sb.append("</table_def>\n");
555     sb.append("<content>\n");
556 
557     for (int i = m_start; i <= m_end; i++) {
558         String id = ((String []) m_results.get(i)) [0];
559         String name = ((String[]) m_results.get(i)) [1];
560         String description = ((String []) m_results.get(i)) [2];
561         String objectType = ((String []) m_results.get(i)) [3];
562 +       String workspace = "";
563 +       if (allWorkspaces) {
564 +           String spaceType = ((String[]) m_results.get(i)) [4];
565 +           String spaceName = ((String[])m_results.get(i)) [5];
566 +           workspace = spaceName + " (" + spaceType + ")";
```

```
567 +       }
568         String  href  = URLFactory.makeURL(id, objectType);
569
570         sb.append("<row>\n");
571         sb.append("<data_href>\n");
572         sb.append("<label>" + XMLUtils.escape(name) + "</label>\n");
573         sb.append("<href>" + href + "</href>\n");
574         sb.append("<id>" + XMLUtils.escape(id) + "</id>");
575         sb.append("</data_href>\n");
576         sb.append("<data>" + XMLUtils.escape(description) + "</data>");
577 +       if  (allWorkspaces) {
578 +           sb.append("<data>" + XMLUtils.escape(workspace) + "</data>");
579 +       }
580         sb.append("</row>\n");
581
582     }
583
584     sb.append("</content>\n");
585     sb.append("</channel>\n");
586
587     return sb.toString () ;
588
589 }
```

Výpis 19: Metoda *DocumentSearch.getXMLResults()*

Výsledek hledání pak v aplikaci vypadá tak, jako na obrázku 9, kde je zeleně označena přidaná část, která se zobrazí jen pokud je zatržena volba pro vyhledávání ve všech agendách.



Obrázek 9: Výsledek vyhledávání

4.3.2.4 Vyhledávací funkce V předchozím odstavci jsem se zmínil o tom, že se volá metoda *getFulltextTable()* třídy *DocumentFulltext*, která má na starost naplnit dočasnou tabulku výsledky vyhledávání. Ještě jsem zůstal dlužen ukázku její implementace. Není to nic složitého, ale pro pořádek zde kód uvedu:

```
199 public static void getFulltextTable(String fulltextQuery, DBBean m_db) throws SQLException {
200     Connection connection;
201     PreparedStatement statement;
202     ResultSet resultSet;
203     long key;
204     int rank;
205
206     // perform fulltext query
207     connection = getConnection();
208     statement = connection.prepareStatement("SELECT [KEY], [RANK] FROM CONTAINSTABLE([
            document_vault], *, ?)");
209     statement.setString(1, fulltextQuery);
210     statement.execute();
211     resultSet = statement.getResultSet();
212
213     // fill temporary table with results
214     while (resultSet.next()) {
215         key = resultSet.getLong(1);
216         rank = resultSet.getInt(2);
217
218         m_db.executeQuery("INSERT INTO tmp_fulltexttable(key, rank) VALUES (" + key + ", " + rank
                + ")");
219     }
220
221     resultSet.close();
222     statement.close();
223     connection.close();
224 }
```

Výpis 20: Metoda *DocumentFulltext.getFulltextTable()*

4.3.2.5 Uživatelská práva pro přístup k dokumentům

Jedním z požadavků bylo respektování uživatelských práv. Project.net sice nedovolí uživateli zobrazit dokumenty, na něž nemá právo, ale havaruje běhovou výjimkou, což není žádoucí.

Dokonce už samotná informace o existenci nějakého dokumentu (název a popis ve výsledcích hledání) může být problém, nehledě na to, že by se dalo správnou volbou dotazů usuzovat na to, co je uvnitř dokumentu. Takže se vyhledávání musí přizpůsobit tak, aby se nezobrazily dokumenty k nimž nemá uživatel právo.

To řeší následující klauzule, která se při fulltextovém vyhledávání vkládá do dotazu:

```
1 WHERE pn_space_view.space_id IN (
2    SELECT space_id FROM pn_space_has_person WHERE person_id = ...
3 )
```

Přitom ID aktuálně přihlášeného uživatele – *person_id* – se získá jako:

```
1 SessionManager.getUser().getID()
```

Tím je zajištěno, že uživatel uvidí jen ty dokumenty, ke kterým má přístup.

4.3.2.6 Označení verze aplikace

V souboru *version.properties* jsem ještě upravil číslo verze, aby bylo patrné, že se jedná o upravenou verzi aplikace:

```
25 product.version.codename=v8.4_fulltext
```

Výpis 21: Soubor core/config/version.properties

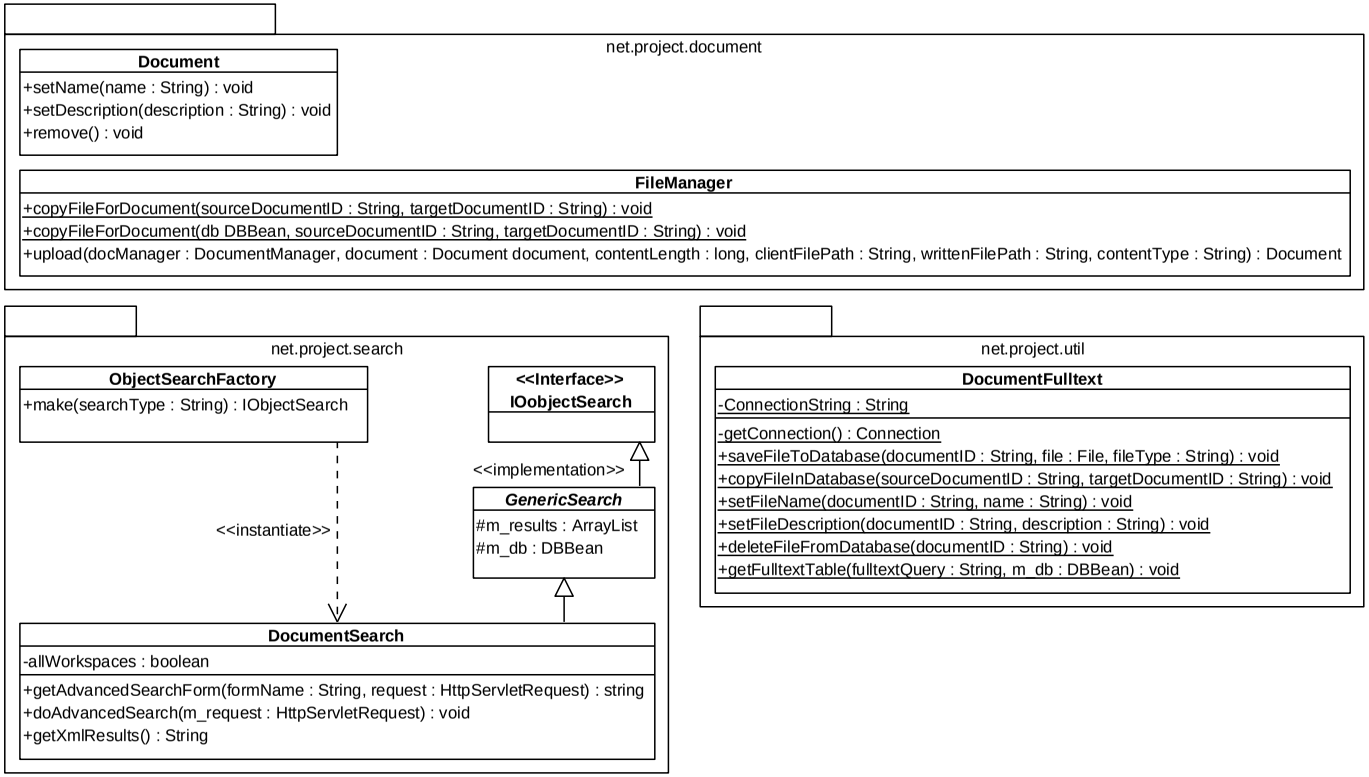
4.4 Popis úprav aplikace v jazyce UML

Pro názornost přidávám ještě UML diagramy, které snad pomohou udělat si obrázek o tom, jak vyhledávání funguje.

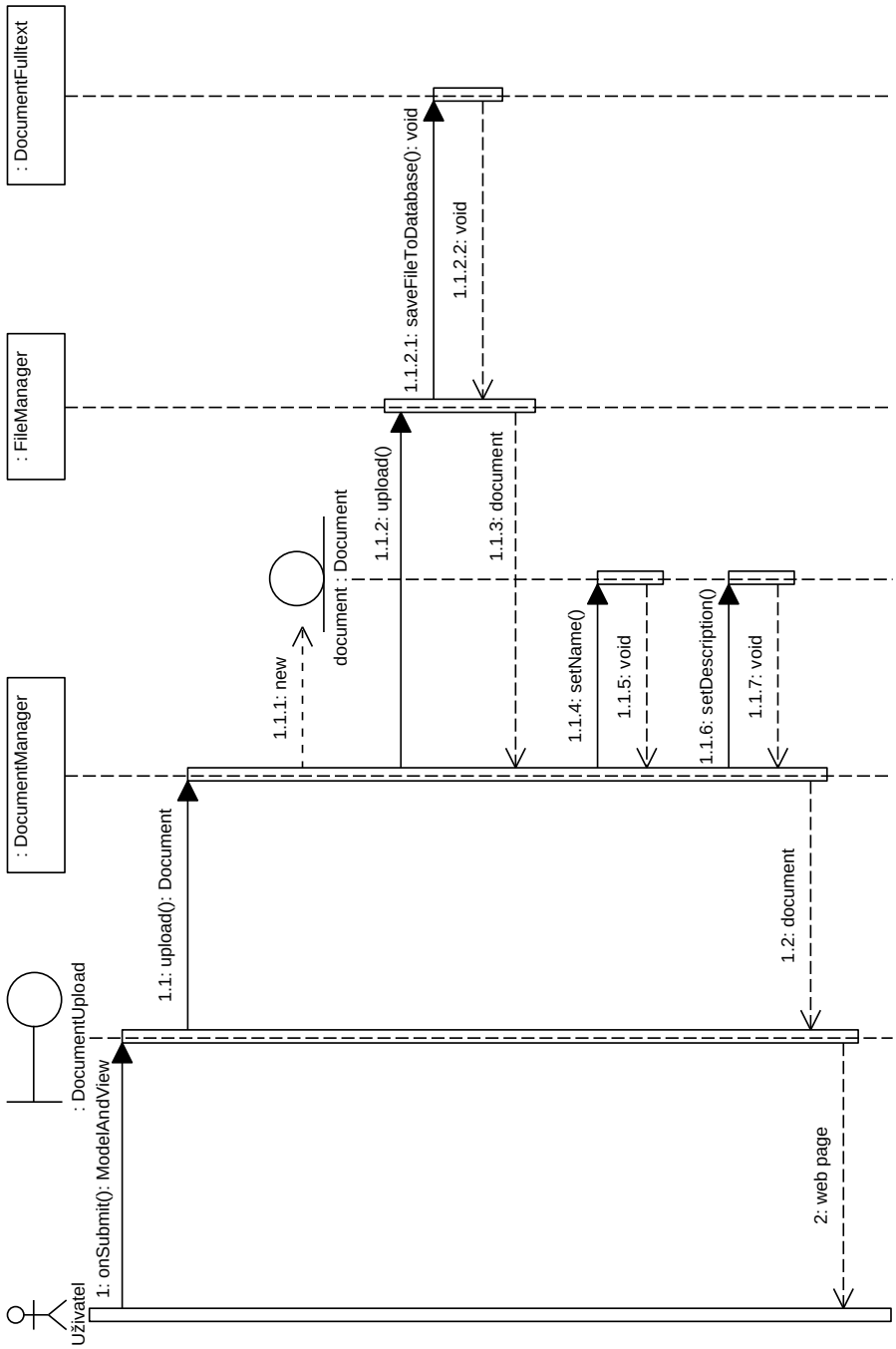
Obrázek 10 je diagram tříd dotčených úpravami. Je na něm vidět rozdělení tříd do balíků a vztahy mezi třídami v balíku *search*. Jedná se o použití návrhového vzoru factory. Aplikace teprve až v průběhu vyhledávání vybere třídu, jejíž instanci má vytvořit, a to podle toho, jaký typ objektu chce zrovna uživatel hledat.

Na obrázku 11 je sekvenční diagram akce přidání nového dokumentu.

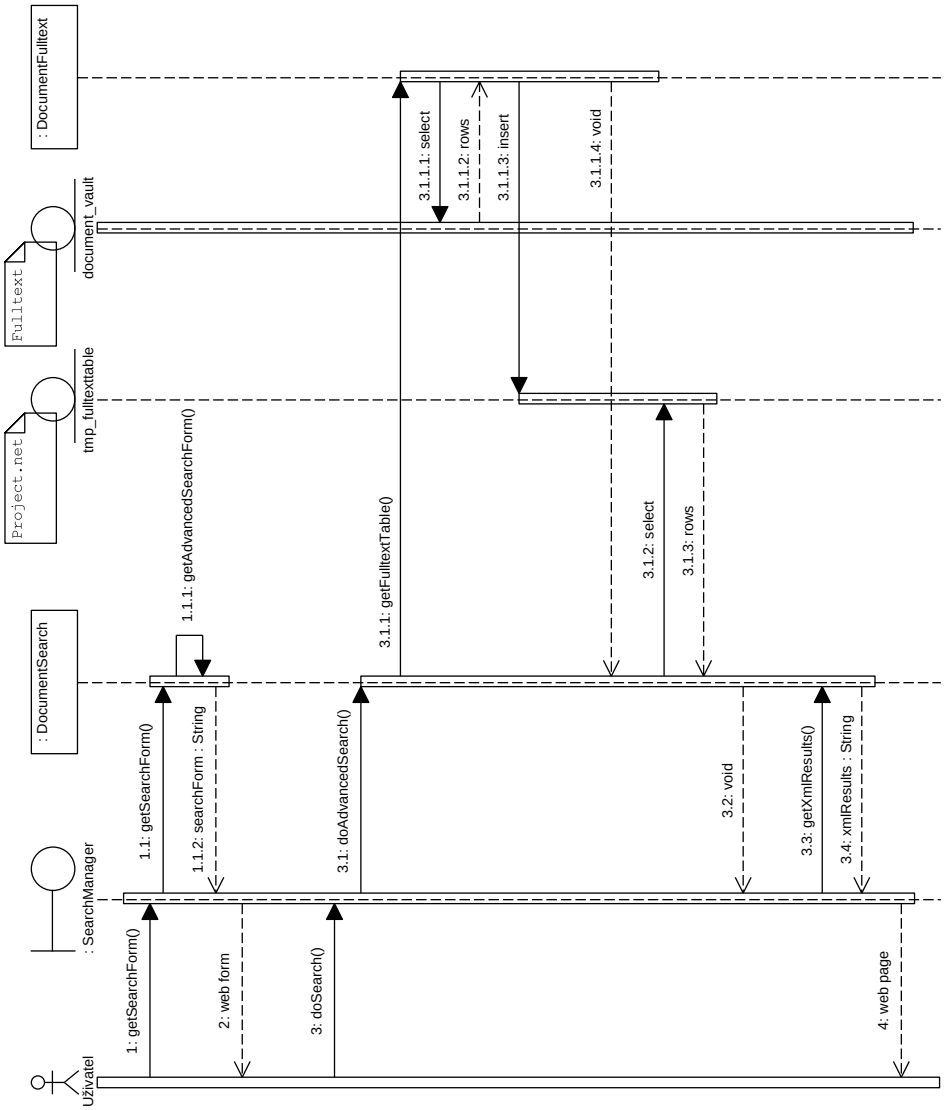
Obrázek 12 ukazuje sekvenční diagram vyhledání dokumentů. Je vidět, že hledání je rozděleno do několika částí – nejprve se zobrazí vyhledávací formulář, pak se zavolá hledání, které napřed naplní dočasnou tabulku výsledků fulltextového hledání v databázi Project.net, a teprve potom vyhledává v nalezených dokumentech podle dalších kritérií, např. uživatelských práv. Výsledky hledání jsou uloženy v objektu DocumentSearch, a později na požádání vráceny jako XML data.



Obrázek 10: Třídní diagram upravovaných částí aplikace



Obrázek 11: Sekvenční diagram přidání dokumentu

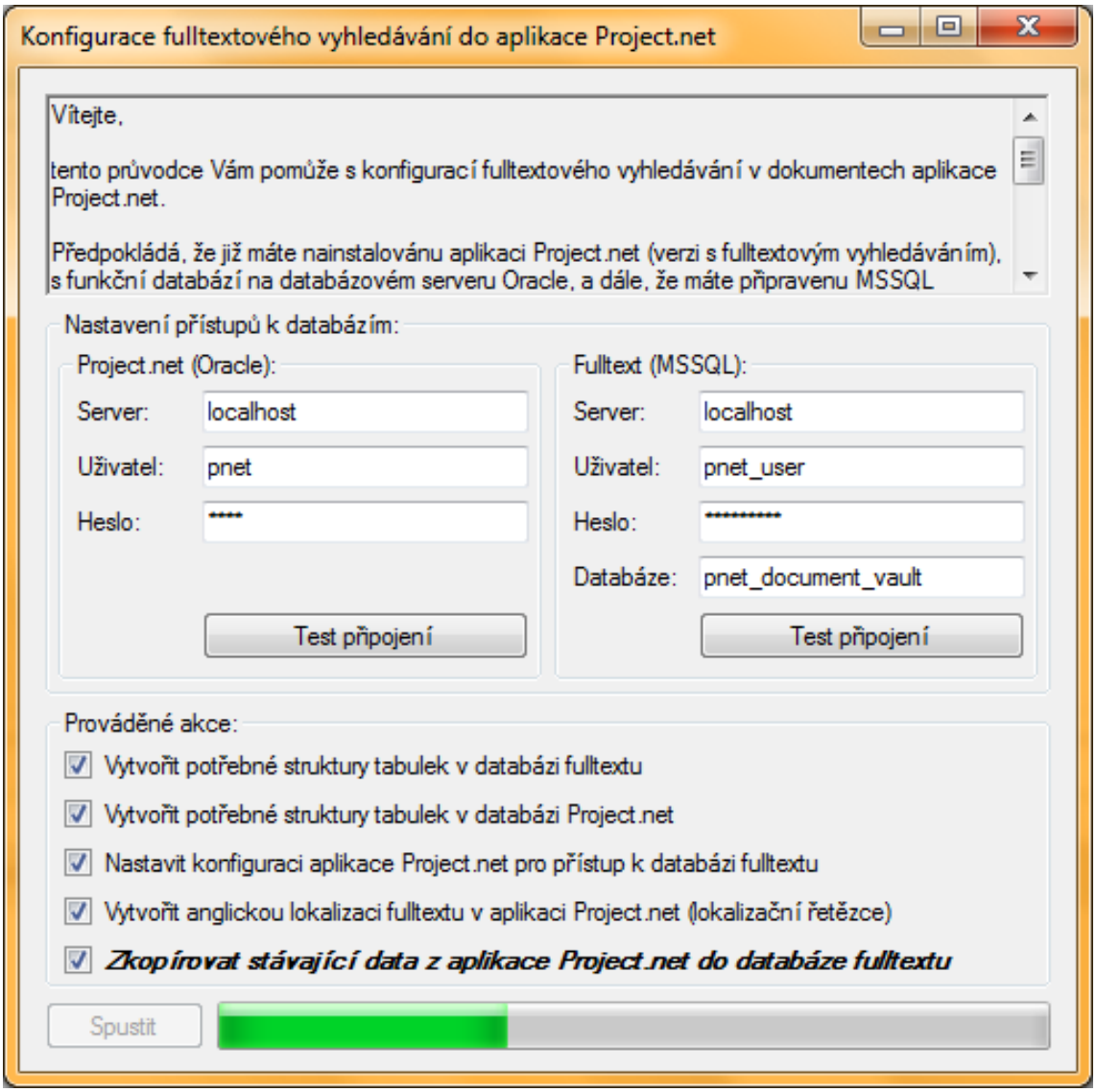


Obrázek 12: Sekvenční diagram vyhledávání

4.5 Program pro automatickou konfiguraci Project.net

Protože úvodní instalace vyhledávání se skládá z několika složitějších kroků, vytvořil jsem program, který tuto práci zjednoduší. Program je psán v jazyce C# na platfomě .NET. Základní obrazovka programu je vidět na obrázku 13.

Stačí vyplnit přihlašovací údaje k databázím, a program zbytek zařídí. V následující části vysvětluji jak program použít, co znamenají jednotlivé akce, a proč jsou důležité.



Obrázek 13: Konfigurační program

4.5.1 Instalační akce

Nejprve je třeba si připravit databáze a vytvořit v nich uživatele. Databázi Oracle máme připravenou již od instalace Project.net v kapitole 2.2.2.3.

Pro fulltext založíme novou MSSQL databázi a uživatele, který má právo vytvářet tabulky a fulltextové katalogy a indexy.

Vyplníme údaje. Pomocí tlačítek *Test připojení* můžeme vyzkoušet, jestli jsou údaje správné. Nakonec tlačítkem *Spustit* zahájíme požadované akce, a pokud je vše v pořádku, za pár okamžiků bude vyhledávání správně fungovat.

4.5.1.1 Vytvořit potřebné struktury tabulek v databázi fulltextu Tato funkce vytvoří tabulku pro fulltext, a potřebné indexy, jak je zmíněno v kapitole 4.2.

```
1  CREATE TABLE [dbo].[document_vault](
2      [document_id] [numeric](20, 0) NOT NULL,
3      [name] [nvarchar](255) NOT NULL,
4      [description] [nvarchar](4000) NOT NULL,
5      [data] [varbinary](max) NOT NULL,
6      [type] [nvarchar](10) NOT NULL,
7      CONSTRAINT [PK_document_vault] PRIMARY KEY CLUSTERED
8      (
9          [document_id] ASC
10     ) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
           ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
11 ) ON [PRIMARY];
12
13 CREATE FULLTEXT CATALOG [pnet_fulltext_catalog] WITH ACCENT_SENSITIVITY = OFF AS
       DEFAULT AUTHORIZATION [dbo];
14 ALTER FULLTEXT INDEX ON [dbo].[document_vault] ADD ([data] TYPE COLUMN [type]
       LANGUAGE [Neutral]);
15 ALTER FULLTEXT INDEX ON [dbo].[document_vault] ADD ([description] LANGUAGE [Neutral]);
16 ALTER FULLTEXT INDEX ON [dbo].[document_vault] ADD ([name] LANGUAGE [Neutral]);
17 ALTER FULLTEXT INDEX ON [dbo].[document_vault] ENABLE;
```

Výpis 22: Vytvářecí dotazy pro databázi fulltextu

4.5.1.2 Vytvořit potřebné struktury tabulek v databázi Project.net

Vytvoří tabulku pro ukládání dočasných výsledků hledání.

```
1 CREATE GLOBAL TEMPORARY TABLE tmp_fulltexttable (
2     key         NUMBER(20,0) NOT NULL,
3     rank        NUMBER(4,0) NOT NULL,
4     CONSTRAINT tmp_fulltexttable_pk PRIMARY KEY (key)
5 )
```

Výpis 23: Vytvářecí dotaz pro databázi Project.net

V databázi Oracle fungují dočasné tabulky tak, že jsou viditelné pro všechny instance, ale každá transakce nebo relace (existují 2 typy dočasných tabulek) má vlastní datový soubor. Takže se struktura vytvoří jen jednou, ale najednou s tabulkou může pracovat několik transakcí, přičemž každá vidí pouze svá data.

V jiných DBMS, např. v MySQL, je existence dočasných tabulek omezená jen na konkrétní spojení, a je tedy nutné je vždy vytvářet znovu. Řešení použité v Oracle, má tedy výkonnostní výhodu, neboť se ušetří čas potřebný pro vytváření a rušení datových struktur.

4.5.1.3 Nastavit konfiguraci aplikace Project.net pro přístup k databázi fulltextu

Tato volba v podstatě jen nastaví Connection String pro MSSQL databázi do properties Project.net, tak aby aplikace měla dostupné přístupové údaje k databázi a mohla se k ní při vyhledávání připojit.

4.5.1.4 Vytvořit anglickou lokalizaci fulltextu v aplikaci Project.net

Tato akce je důležitá, protože aplikace má podporu pro lokalizaci, a namísto textových řetězců se v uživatelském rozhraní používají symbolické názvy pro jednotlivé řetězce. Aplikace pak zobrazuje tyto texty podle aktuálně uživatelem vybraného jazyka.

Funkce tedy vloží do databáze řetězce pro angličtinu, aby aplikace nezobrazovala chybovou hlášku, že řetězce chybí.

4.5.1.5 Zkopírovat stávající data z aplikace Project.net do databáze fulltextu

Tato volba se hodí v případě, že jsou v aplikaci Project.net nějaké dokumenty ještě před instalací fulltextového vyhledávání.

Funkce projde databázi, najde všechny dokumenty, sestaví cestu k souborům, a nakonec data z těchto souborů nahraje do fulltextové databáze. Pokud je v aplikaci hodně dokumentů, může tato akce trvat velmi dlouho.

4.6 Zhodnocení splnění požadavků

Když se podívám znovu na požadavky definované v kapitole 4.1, vidím, že se mi je podařilo všechny splnit:

- **Indexace binárních formátů** – Požadavky byly splněny téměř automaticky díky schopnostem MSSQL Serveru a jeho rozšíření o další souborové formáty.
- **Jednoduché nastavení a obsluha** – Pro uživatele obsluha nemůže být jednodušší, byly využity standardní formuláře aplikace, na které jsou uživatelé zvyklí.

  Pokud jde o instalaci, ta je zjednodušena speciální aplikací, a myslím si, že kdo dokázal nainstalovat Project.net, zvládne i instalaci fulltextu.
- **Možnost hledání napříč aplikací** – Byla doplněna tato možnost, a funguje přesně podle zadání.
- **Řazení výsledků dle relevance** – Dokumenty jsou uživateli zobrazeny v pořadí, v jakém je hodnotí MSSQL.
- **Respektování uživatelských práv** – Vyhledávání nezobrazí uživateli dokumenty, ke kterým se nemůže dostat jinak.
- **Minimální zásahy do existujících souborů** – Kromě jednoho souboru jsou všechny změny jednořádkové, jedná se jen o volání funkcí, jejichž definice jsou umístěny v jednom samostatném souboru.

5 Závěr

Na této práci hodnotím významně to, že se mi podařilo spojit různé technologie – databáze Oracle a Microsoft SQL Server, a vytvořit dohromady funkční celek.

Ověřil jsem si, že programování v Javě, resp. C# je mnohem jednodušší, než tomu je v případě PHP, se kterým mám bohaté zkušenosti z dřívější doby, mj. ze své bakalářské práce [11].

Programování webového rozhraní v JSP považuji přinejmenším za těžkopádné, a bez využití nějakého pořádného frameworku za téměř nepoužitelné.

Pokud bych měl vybrat svého favorita mezi databázovými servery MSSQL a Oracle, obojí jsou kvalitní databázové nástroje, ale produkt Microsoftu hodnotím výše, hlavně kvůli aplikaci SQL Server Management Studio, která dokáže značně ulehčit práci. Jak Oraclu, tak i MySQL chybí nástroj podobných kvalit.

Fulltextové vyhledávání v MSSQL Serveru považuji za vynikající, můžu srovnávat s fulltextem v MySQL, a ten je oproti němu velmi omezený, hlavně po stránce jazykových funkcí. Díky experimentům a měřením, která jsem během této práce provedl, se jasně ukázalo, jak je fulltextové vyhledávání rychlé, a pro větší objemy dat nesrovnatelně výkonnější, než prosté LIKE hledání.

Co se týče dalšího využití výsledků práce, myslím, že užitečnost a rychlost hledání stojí za tu trochu námahy s instalací, a věřím, že hledání pomůže řadě uživatelů zorientovat se ve velkém množství dokumentů. S vedoucím práce se domlouváme na nasazení na některých jeho projektech.

Pokud jde o možná rozšíření aplikace, hodilo by se doplnit tzv. „snippety“, tedy útržky textu vytažené z dokumentu, kde je vidět hledaná fráze v širším kontextu. To uživatelům usnadňuje udělat si přehled o obsahu dokumentu. Bohužel MSSQL tuto funkci nemá.

Dále by se hodilo aplikaci zrevidovat, jakmile budou dostupné nové jazykové funkce pro MSSQL v češtině, ať už seznam stop slov, skloňování nebo thesaurus.

Chcete-li vidět funkční hledání v akci, bez nutnosti všechno instalovat a nastavovat, což uznávám může být složité, doporučuji si prohlédnout video, které je na CD, jež je přílohou této práce.

6 Reference

[1] *Project.net — Open Source Web-based Project Management Software, Project Portfolio Management Software* [online]. ©2010 [cit. 2010-03-22].

Dostupné z WWW: <http://www.project.net/>.

[2] *DevelopmentInformation - Project.net Community Development - Trac* [online]. [cit. 2010-03-22].

Dostupné z WWW: <http://dev.project.net/trac/pnet-community/wiki/DeveloperInformation>.

[3] Data access object In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 29 July 2004, 20 February 2010 [cit. 2010-03-22].

Dostupné z WWW: <http://en.wikipedia.org/wiki/Data_access_object>.

[4] Hypertext Transfer Protocol In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 23. 9. 2004, 7. 2. 2010 [cit. 2010-03-28].

Dostupné z WWW: <http://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol>.

[5] *HTTP/1.1: Status Code Definitions* (část z RFC2616) [online]. 2004/09/01 [cit. 2010-03-28].

Dostupné z WWW: <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>.

[6] *HTML & CSS - W3C* [online]. ©2010 [cit. 2010-03-23].

Dostupné z WWW: <http://www.w3.org/standards/webdesign/htmlcss>.

[7] Java (programovací jazyk) In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 10. 1. 2008, 21. 11. 2009 [cit. 2010-03-28].

Dostupné z WWW: <http://cs.wikipedia.org/wiki/Java_(programovac%C3%AD_jazyk)>.

[8] JavaServer Pages In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 14 September 2009, 23 March 2010 [cit. 2010-03-28].

Dostupné z WWW: <http://en.wikipedia.org/wiki/Java_Server_Pages>.

[9] KUMAR, Deepak. *Deepak Kumar's Home Page* [online]. 2010 [cit. 2010-03-28]. Learning JSP.

Dostupné z WWW: <http://mainline.brynmawr.edu/~dkumar/JSP/>.

[10] Microsoft SQL Server In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida): Wikipedia Foundation, 1 November 2002, 30 March 2010 [cit. 2010-04-08].

Dostupné z WWW: <http://en.wikipedia.org/wiki/Microsoft_SQL_Server>.

[11] ŠUSTEK, Martin. *Agregátor nabídky zboží elektronických obchodů*.

Ostrava, 2008. 48 s. Bakalářská práce. VŠB - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky.

[12] How to: Alter the List of Registered Word Breakers and Filters (Transact SQL) In *Microsoft TechNet* [online]. [cit. 2010-04-08].

Dostupné z WWW: <http://technet.microsoft.com/en-us/library/dd207002.aspx>.

[13] Querying SQL Server Using Full-Text Search IN *SQL Server 2008 Books Online* [online]. [cit. 2010-04-10].

Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms142559.aspx>.

7 Přílohy

Součástí této práce je i CD nosič s následujícím obsahem:

I. **PDF verze textu** – Přesná kopie tohoto textu ve formátu PDF.

II. **Kopie zadání práce** – Kopie zadání práce ve formátu PDF.

III. **Uživatelská příručka vyhledávání** – Návod pro použití fulltextového vyhledávání pro uživatele, obsahuje i nápovědu k syntaxi pro zadávání vyhledávacích dotazů.

IV. **Ukázkové video aplikace** – Demonstrace vyhledávání v aplikaci a ukázka použití programu pro konfiguraci vyhledávání.

V. **Zdrojové kódy aplikace** – Kopie zdrojových kódů upravené verze aplikace.